Malware memory analysis for non-specialists

Investigating publicly available memory images for Prolaco and SpyEye

R. Carbone
Certified Forensic Hacking Investigator (EC-Council)
Certified Incident Handler (SANS)
DRDC Valcartier

Defence Research and Development Canada – Valcartier

Technical Memorandum
DRDC Valcartier TM 2013-155
October 2013

Abstract

This technical memorandum examines how an investigator can analyse an infected Windows memory dump. The author investigates how to carry out such an analysis using Volatility and other investigative tools, including data carving utilities and anti-virus scanners. Volatility is a popular and evolving open source-based memory analysis framework upon which the author has proposed a memory-specific methodology for aiding fellow novice memory analysts. The author examines how Volatility can be used to find evidence and indicators of infection. This technical memorandum is the second in a series concerning Windows malware-based memory analysis. This current work examines two memory images infected with Prolaco and SpyEye, respectively.

Résumé

Ce mémorandum technique examine comment un investigateur peut analyser une image mémoire d'une machine Windows infectée. L'auteur investigue les techniques d'analyse utilisant Volatility et d'autres outils tels que les utilitaires de récupération de données et les scanneurs anti-virus. Volatility est un cadre populaire d'analyse de mémoire en source libre sur lequel l'auteur s'appuie pour proposer une méthodologie spécifique à la mémoire pour aider ses collègues analystes novices. L'auteur examine comment Volatility peut être utilisé pour trouver des preuves et des indicateurs d'infection. Ce mémorandum technique est le deuxième d'une série visant la découverte de maliciel par le biais d'une analyse de la mémoire. Le présent travail examine deux images mémoires infectées, respectivement, par Prolaco et SpyEye.

This page intentionally left blank.

Executive summary

Malware memory analysis for non-specialists: Investigating publicly available memory images for Prolaco and SpyEye

Carbone, R.; DRDC Valcartier TM 2013-155; Defence Research and Development Canada – Valcartier; October 2013.

While memory analysis has largely been carried out by software reverse engineers and malware analysts, the advent of memory analysis-based forensic frameworks such as Volatility, has made it possible for non-memory specialists to engage in the forensic analysis of malware-infected memory images. By combining Volatility, data carving utilities and anti-virus scanners, novice analysts have all the necessary tools required for conducting memory-based investigations.

The author's primary objective is to demonstrate through tutorials how investigators can conduct meaningful memory-based investigations on their own.

This technical memorandum examines two memory images; the first infected with Prolaco and the second with SpyEye, in order to build a compendium of tutorials that can be used by the Canadian Armed Forces and our partners as a basis for conducting their own investigations. This work is the second in a series that examines various Windows-based malware infected memory images. The first report in this series, TM 2013-018, examined the Zeus Trojan horse. It is hoped that these documents will serve as a learning guide.

Although others have engaged in the analysis of some of these publicly available memory images, the author is of the opinion that these analyses are insufficient for use as a learning guide. Specifically, these analyses are either too limited in their investigative scope or report too little information to be of much use to budding memory analysts. Moreover, many of the analyses leave the reader asking more questions than when he began, due to their overall lack of a comprehensive investigative context. Thus, the author has strived to ensure that his investigative actions and lines of inquiry were well documented, even if some portions of a given investigation were unsuccessful, in order to ensure that the investigative context used was coherent.

This work was carried out over a period of several months as part of the Live Computer Forensics project, an agreement between DRDC Valcartier and the RCMP (SRE-09-015, 31XF20).

The results of this project will also be of great interest to the Canadian Forces Network Operations Centre (CFNOC), the RCMP's Integrated Technological Crime Unit (ITCU), the Sûreté du Québec and other cyber investigation teams.

Sommaire

Malware memory analysis for non-specialists: Investigating publicly available memory images for Prolaco and SpyEye

Carbone, R. ; DRDC Valcartier TM 2013-155 ; Recherche et développement pour la défense Canada – Valcartier; octobre 2013.

Bien que l'analyse de la mémoire ait été principalement effectuée jusqu'à présent par les rétro-ingénieurs logiciels et les analystes de maliciel, les avancées des cadres d'analyse de la mémoire, tel que Volatility, permettent maintenant aux non-spécialistes de la mémoire d'effectuer des analyses d'image mémoire de machines infectées par des maliciels. En combinant Volatility, les outils de récupération de données et les scanneurs anti-virus, les analystes novices possèdent tous les outils requis pour investiguer une image mémoire.

L'objectif premier de l'auteur est de démontrer, par le biais d'un tutoriel, comment un investigateur peut réaliser une analyse de la mémoire par lui-même.

Ce mémorandum technique examine deux images mémoires infectées par Prolaco et SpyEye, deux maliciels connus, pour monter un ensemble de tutoriels qui pourront être utilisés par les Forces Armées canadiennes et nos partenaires pour faire leurs propres investigations. Ce travail est le deuxième d'une série visant la découverte de maliciel par le biais d'une analyse de la mémoire d'une machine Windows infectée. Le premier rapport de cette série, TM 2013-018, examinait le cheval de Troie Zeus. Nous espérons que ces documents serviront de guide d'apprentissage.

Bien que d'autres aient mentionné avoir effectué l'analyse de ces images mémoires publiques, l'auteur croit que ces analyses ne sont pas assez détaillées pour servir de guide d'apprentissage. Spécifiquement, ces analyses sont soit trop limitées dans ce qu'elle investigue ou ne donnent pas assez de détails pour être complètement utiles. De plus, plusieurs de ces analyses font que le lecteur a, en bout de ligne, plus de questions que de réponses étant donné le peu de détails approfondis sur le contexte de l'investigation. L'auteur a donc déployé tous les efforts pour s'assurer que toutes les actions et les champs d'enquête sont bien documentés et cohérents dans le contexte, même si certains essais étaient infructueux.

Ce travail fut réalisé sur une période de plusieurs mois dans le cadre du projet "Live Computer Forensics" qui est une entente entre RDDC Valcartier et la GRC (SRE-09-015, 31XF20).

Les résultats de ce projet seront également d'un grand intérêt pour le Centre d'opérations des réseaux des Forces canadiennes (CORFC), le Groupe intégré de la criminalité technologique (GICT) de la GRC, la Sûreté du Québec, ainsi que d'autres équipes d'enquêtes cybernétiques.

Table of contents

List of tables

Acknowledgements

The author would like to thank Mr. Francois Rheaume, Defence Scientist, for peer reviewing this text and providing helpful comments to improve it. Furthermore, the author extends his gratitude to Mr. Martin Salois, Defence Scientist, for translating portions of this text.

Disclaimer policy

It must be understood from the outset that this technical memorandum examines computer malware and that handling virulent software is not without risk. As such, the reader should ensure that he has taken all the necessary precautions to avoid infecting his own computer system and those around him, whether on a corporate network or isolated system.

The reader should neither construe nor interpret the work described herein by the author as an endorsement of the aforementioned techniques and capacities as suitable for any specific purpose, construed, implied or otherwise. Moreover, the author does not endorse the specific use of one specific anti-virus product, the use of Volatility or any data carving technology. Many similar software tools, utilities and scanners exist beyond those used herein. They may be commercial, free or open source in nature and as such, the onus is on the reader to determine which software best suits his specific needs. While the author felt most comfortable working from within a Linux environment, the author does not specifically recommend the use of such a system for the reader. Instead, the reader should use the environment in which he is most comfortable.

Furthermore, the author of this technical memorandum absolves himself in all ways conceivable with respect to how the reader may use, interpret or construe this technical memorandum. The author assumes absolutely no liability or responsibility, implied or explicit. Moreover, the onus is on the reader to be appropriately equipped and knowledgeable in the application of digital forensics. Due to the offensive nature of computer malware, the author is no way responsible for the reader's use of any malware, whether examined herein or otherwise, in any offensive or defensive nature against any other entity, or even against the reader himself, for any purposes whatsoever, for any construed reasons.

Finally, the author and the Government of Canada are henceforth absolved of all wrongdoing, whether intentional, unintentional, construed or misunderstood on the part of the reader. If the reader does not agree to these terms, then his copy of this technical memorandum should be destroyed. Only if the reader agrees to these terms should he or she continue in reading it beyond this point. It is further assumed by all participants that if the reader has not read said Disclaimer upon reading this technical memorandum and has acted upon its contents, then the reader assumes all responsibility for any repercussions that may result from the information and data contained herein.

Requirements, assumptions and exclusions

The author assumes that the reader is altogether familiar with digital forensics and the various techniques and methodologies associated therein. This technical memorandum is not an introduction to digital forensics or to said techniques and methodologies. However, it will endeavour to ensure that the reader can carry out his own forensic analysis of computer memory images suspected of malware infection.

The experimentation conducted throughout this technical memorandum has been carried out atop a Fedora Core 18 64-bit Linux operating system. Six different anti-virus scanners were used throughout this investigation. They include, in alphabetical order, AVG, Avast, BitDefender, ClamAV, FRISK F-Prot and McAfee command line scanners. As for data carving tools and utilities, the author used Photorec, part of the Testdisk suite of data recovery tools.

The reader is required to have permission to use these tools on his computer system or network. Use of these tools and the analysis of virulent software always carry some inherent risk that must be adequately and securely managed and mitigated.

An in-depth study of memory analysis techniques is outside the scope of this work, as it requires a comprehensive study of Windows operating system internals and software reverse engineering techniques, both of which are difficult subjects to approach. Instead, this work should be considered as a guide to using the Volatility memory analysis framework with respect to malware infection.

When examining files dumped using the *malfind*, *memdump*, *procexedump* and *procmemdump* plugins the use of the terms processes, memory sample files and memory dump files are used interchangeably.

Finally, the use of masculine is employed throughout this text to simplify it.

Target audience

The targeted audience for this technical memorandum is the computer forensic investigator who assesses suspect computer memory images for evidence of infection. Although computer memory analysis is a new field within the realm of digital forensics, there are those who have been conducting malware analysis and software reverse engineering for years, long before it came to attention of most practitioners. Those seasoned veterans are aptly skilled and their abilities took years to develop. Thus, a framework such as Volatility, while capable of providing insight to novices, is all the more capable in the hands of experts.

The author has written this technical memorandum for others who, like himself, are required from time to time to conduct memory malware assessments and investigations. However, the author, like many others, is not seasoned enough to take full advantage of Volatility's capabilities. As such, this technical memorandum combines both traditional forensic investigative techniques, coupled with Volatility's non-expert plugins, in order to develop an investigative how-to for non-memory experts.

1 Background

1.1 Objective

The objective of this technical memorandum is to examine how a computer forensic investigator, without specialised computer memory or software reverse engineering knowledge, can successfully investigate a memory image suspected of infection. More specifically, this document provides a methodological approach novice memory analysts can use to investigate suspected memory images.

The work carried out herein is based on two publicly available memory images, specifically Prolaco and SpyEye. This document, the second in a series of many, examines the investigative techniques necessary for a novice to conduct such memory analyses on his own. The first report on this topic written by the author examined the Zeus Trojan Horse, found in TM 2013-018 [22].

Ultimately, these reports will provide a methodological and foundational framework that novice memory analysts and experienced investigators alike can rely on for guidance.

1.2 Why write new tutorials?

The purpose of writing new tutorials was addressed in the first report of this series. [22]

1.3 Infected memory image information

The infected Prolaco and SpyEye memory dump files examined herein were procured from the following location: http://code.google.com/p/volatility/wiki/PublicMemoryImages. Their SHA1 hashes, in uncompressed, form are as follows:

Table 1: Infected memory image metadata.

Memory image name	Size (MiB)	SHA1 hash value
prolaco.vmem	128 (exactly)	85263aec5f1d4c4f5d18a3ba88036602ac06db5
spyeye.vmem	512 (exactly)	5f94d263dfbabaf4373c21e0ff4ba0c4ca0e0921

1.4 Data carving

An in-depth examination of data carving can be found in two memorandums written by the author, specifically [22][23].

1.5 Malware and anti-virus scanners

1.5.1 Specifics

An examination of malware and anti-virus scanner specifics can be found in [22].

1.5.2 Caveat

An analysis concerning the caveats of using malware and anti-virus scanners was conducted in [22].

1.6 Detailed list of software tools used

1.6.1 Anti-virus scanners

This memorandum makes use of six anti-virus scanners, the same six used in [22] because they continue to represent a diverse cross-section of various detection mechanisms necessary to detect various malware. Each scanner was last updated August 5, 2013, the date upon which the analysis was carried out herein. Scanner specifics are listed in the following table:

Table 2: List of anti-virus scanners and their command line parameters.

Anti-virus scanner	Command line parameters
AVG 2013 command line scanner version 13.0.3114	-H -P -p
Avast v.1.3.0 command line scanner	-c
BitDefender for Unices v7.90123 Linux-amd64 scanner command line	No parameters used
ClamAV 0.97.8/17633/Tue Aug 6 11:04:00 2013 command line scanner	--detect-pua=yes --detect-broken=yes -r
FRISK F-Prot version 6.3.3.5015 command line scanner	-u 4 -s 4 -z 10 --adware --applications
McAfee VirusScan for Linux64 Version 6.0.3.356 command line scanner	--RECURSIVE --ANALYZE --MANALYZE --MIME --PANALYZE --UNZIP --VERBOSE

1.6.2 Data carving

Photorec was used for data carving. The specifics concerning the version and program settings are examined in [22].

1.6.3 Volatility

An examination of Volatility, its capabilities, main authors and contributors is found in [22].

A list of Windows-specific plugins currently supported by this version Volatility is described in Annex B.

1.7 Investigative methodology

The investigative methodology has evolved slightly changed since it was first proposed when examining and analysing the Zeus-infected computer memory image. The original methodology is found in [22].

The following modifications pertain to Part 6 of the methodology. It was adapted so that it can be applied to infections not relying on the Windows registry or which leave little trace of itself in memory. The proposed additions are as follows:

Part 6:

1. Using *strings*-based extraction, find all 7, 8, 16 and 32-bit strings for the memory image and all suspect or infected process-based dumps. Using established filters or wordlists, determine which strings are likely applicable to the current infection, investigation or analysis. Wordlists may be created as needed based on the underlying context. Determining which strings are applicable to the current infection is largely manual in nature although wordlists should considerably reduce the output to be analysed.

2. Equipped with a list of applicable strings, determine their context with respect to the current suspect infection and then if possible establish the presence and behaviour of the malware.

2 Memory investigation and analysis of Prolaco

2.1 Background

This analysis examines a memory image suspected of harbouring the Prolaco worm, as based on the methodology put forward in Section 1.7. Little useful information could be found concerning the technical details of this infection, contrary to the plethora of information available for the Zeus infection examined in TM 2013-018 [22]. What was found in [2][3] and [4] did not provide any clear indicators of compromise concerning this specific malware infection.

2.2 Preliminary investigative steps

The steps examined in this subsection should be considered as preliminary investigative steps necessary for examining a potentially infected memory image.

2.2.1 Safeguard the memory image

The memory image *prolaco.vmem* was set to immutable atop an Ext4-based filesystem. The command used to perform this, carried out as the root user, was:

```
$ sudo chattr +i prolaco.vmem
```

This results in a memory image that can no longer be modified, even by the root user. This is to prevent accidental modifications from occurring to this file.

2.2.2 Preliminary anti-virus scanning results

Scanning only the memory image itself with the six scanners outlined in Section 1.6.1, it was determined that, unlike the Zeus infected memory image (see [22] for details) none of them identified the memory image as infected.

Even though no infection was found, all scanner results were saved for possible future use.

2.2.3 Data carving and file hashing

Photorec succeeded in recovering 804 files carved from the Prolaco memory image as per the recommended Photorec settings put forward in Section 1.6.2. Twenty-three duplicate files were found, thereby leaving 781 unique files recovered. Of those 804 recovered files, 238 were identified as PE-based files. Of those, 126 were identified as Windows 32-bit DLLs, while 112 were identified as standard Windows 32-bit PEs and device drivers. No 64-bit PE-based files were identified nor were any UPX-based files detected.

Other file types were detected but were of no immediate use. However, their types were recorded and saved for possible future use within this analysis.

All recovered files were SHA1-hashed and then validated against NSRL hash-set 2.40 (March 2013). Results were stored for future use. Two SHA1 hashes were confirmed as matches against the NSRL hash-set. Information including NSRL matching filenames can be found in Annex C.1.

Finally, CTPH-based hashing (fuzzy hashing) was conducted using the *ssdeep* tool against the carved data files and stored for future use.

2.2.4 Anti-virus scanning and file hashing results for carved data files

Using the six scanners and combining their output through UNIX command line processing tools (e.g. *cat*, *sort*, *find*, *tr*, *strings*, *awk*, *grep*, *uniq*, etc.), two matches were established between scanners AVG and ClamAV. These matches were files *f0139704.exe* and *f0235672.dll* which are found indicated accordingly in Annex A.1. A match occurs when two or more scanners detect a file as infected or possibly malicious.

Of the six scanners, only three were capable of detecting one or more potential infections. These scanners included AVG, BitDefender and ClamAV. Interestingly, none of these potentially infected files were detected as the Prolaco worm or anything else remotely resembling the name of this infection. The following table details the two files that were matched between the AVG and ClamAV scanners.

Table 3: Matching of potentially infected carved data file vs. scanner (Prolaco)

Potentially infected file	Detecting scanner
f0139704.exe	AVG
	ClamAV
f0235672.dll	AVG
	ClamAV

Specific logs for each scanner can be found in Annex A.1.

2.3 Volatility analysis

In order to investigate this specific memory image the use and output of various Volatility plugins of assistance to this particular analysis are examined.

2.3.1 Step 1: Determine background information

This step examines the Volatility plugins used to provide background information and context to the memory image.

2.3.1.1 Imageinfo plugin

This Volatility plugin is used to provide basic contextual information about a suspect memory image.

Consider the following output from this plugin, using command "*volatility -f prolaco.vmem imageinfo*":

```
Determining profile based on KDBG search...

          Suggested Profile(s) : WinXPSP2x86, WinXPSP3x86
(Instantiated with WinXPSP2x86)
                     AS Layer1 : JKIA32PagedMemoryPae (Kernel AS)
                     AS Layer2 : FileAddressSpace
(/media/scratch/Report2_SpyEye_Prolaco/Prolaco/prolaco.vmem)
                      PAE type : PAE
                           DTB : 0x319000L
                          KDBG : 0x80544ce0
          Number of Processors : 1
     Image Type (Service Pack) : 2
                KPCR for CPU 0 : 0xffdff000
             KUSER_SHARED_DATA : 0xffdf0000
           Image date and time : 2010-08-11 17:55:09 UTC+0000
     Image local date and time : 2010-08-11 13:55:09 -0400
```

This memory image appears to be running atop a 32-bit Windows XP computer system with Service Pack 2. It is equipped with one PAE-based processor and the memory image is 128 MiB in size (based on the memory image's size determined using *ls -l*). The memory image was captured August 11, 2010 at 13:55:09 EDT.

2.3.1.2 Pslist plugin

The next step is to determine which processes are running within the memory image in order to determine if anything out of the ordinary is immediately visible. The *pslist* plugin provides a detailed process listing. It makes use of virtual memory addresses and offsets.

Consider the following output from this plugin, using command "*volatility -f prolaco.vmem pslist*":

Table 4: Volatility output for the Pslist plugin sorted by PID (Prolaco).

Offset(V)	Name	PID	PPID	Thds	Hnds	Sess	Wow64	Start	Exit
0x810b1660	System	4	0	56	253	------	0		
0xff25a7e0	alg.exe	216	676	6	104	0	0	2010-08-11 06:06:39	
0xff3667e8	VMwareTray.exe	432	1724	1	49	0	0	2010-08-11 06:09:31	
0xff374980	VMwareUser.exe	452	1724	5	176	0	0	2010-08-11 06:09:32	
0x80f94588	wuauclt.exe	468	1028	3	130	0	0	2010-08-11 06:09:37	
0xff2ab020	smss.exe	544	4	3	21	------	0	2010-08-11 06:06:21	
0xff1ecda0	csrss.exe	608	544	11	349	0	0	2010-08-11 06:06:23	
0xff1ec978	winlogon.exe	632	544	19	565	0	0	2010-08-11 06:06:23	

Offset(V)	Name	PID	PPID	Thds	Hnds	Sess	Wow64	Start	Exit
0xff247020	services.exe	676	632	16	269	0	0	2010-08-11 06:06:24	
0xff255020	lsass.exe	688	632	19	341	0	0	2010-08-11 06:06:24	
0xff218230	vmacthlp.exe	844	676	1	24	0	0	2010-08-11 06:06:24	
0x80ff88d8	svchost.exe	856	676	16	198	0	0	2010-08-11 06:06:24	
0xff364310	wscntfy.exe	888	1028	1	27	0	0	2010-08-11 06:06:49	
0xff217560	svchost.exe	936	676	9	256	0	0	2010-08-11 06:06:24	
0x80fbf910	svchost.exe	1028	676	63	1334	0	0	2010-08-11 06:06:24	
0xff38b5f8	TPAutoConnect.e	1084	1968	1	61	0	0	2010-08-11 06:06:52	
0xff22d558	svchost.exe	1088	676	4	75	0	0	2010-08-11 06:06:25	
0xff37a4b0	ImmunityDebugge	1136	1724	2	73	0	0	2010-08-11 16:50:19	
0xff203b80	svchost.exe	1148	676	14	207	0	0	2010-08-11 06:06:26	
0xff1d7da0	spoolsv.exe	1432	676	13	135	0	0	2010-08-11 06:06:26	
0xff1b8b28	vmtoolsd.exe	1668	676	5	219	0	0	2010-08-11 06:06:35	
0xff3865d0	explorer.exe	1724	1708	11	294	0	0	2010-08-11 06:09:29	
0xff1fdc88	VMUpgradeHelper	1788	676	3	97	0	0	2010-08-11 06:06:38	
0xff143b28	TPAutoConnSvc.e	1968	676	5	100	0	0	2010-08-11 06:06:39	

Certain process names are not particularly common, including *alg.exe* and *ImmunityDebugger.exe*. However, the use of Immunity is not necessarily indicative of an infection. Instead, it merely signifies that some individual on this computer system was running a debugger.

VMware-based processes including *vmacthlp.exe*, *VMUpgradeHelper.exe*, *TPAutoConnSvc.exe*, *VMwareTray.exe* and *VMwareUser.exe* should not, in of themselves, raise any specific suspicion.

Possibly suspicious is the presence of process *alg.exe*, the Windows Application Layer Gateway, a process used to establish specific types of connections commonly used for Instant Messaging, RTSP, BitTorrent, SIP and FTP [1]. By itself, this process is not necessarily suspicious but the services that could be used by a potential infection may be using this service to provide a means of establishing an illicit communication. Further analysis using the *psscan* and *connscan* plugins may be of further help.

2.3.1.3 Psscan plugin

The *psscan* plugin uses physical memory addresses and scans memory images for _EPROCESS pool allocations, in contrast to the *pslist* plugin that uses virtual memory addresses and scans for EPROCESS lists. The benefit of using this plugin is that sometimes it can succeed in listing processes that cannot be found using any of the other process listing plugins (i.e., *pslist* and *pstree*).

Consider the following output from this plugin, using command "*volatility -f prolaco.vmem psscan*":

Table 5: Volatility output for the Psscan plugin sorted by PID (Prolaco)

Offset(P)	Name	PID	PPID	PDB	Time created	Time exited
0x01214660	System	4	0	0x00319000		
0x05f027e0	alg.exe	216	676	0x06cc0240	2010-08-11 06:06:39	
0x04be97e8	VMwareTray.exe	432	1724	0x06cc02e0	2010-08-11 06:09:31	
0x04b5a980	VMwareUser.exe	452	1724	0x06cc0300	2010-08-11 06:09:32	
0x010f7588	wuauclt.exe	468	1028	0x06cc0180	2010-08-11 06:09:37	
0x05471020	smss.exe	544	4	0x06cc0020	2010-08-11 06:06:21	
0x066f0da0	csrss.exe	608	544	0x06cc0040	2010-08-11 06:06:23	
0x066f0978	winlogon.exe	632	544	0x06cc0060	2010-08-11 06:06:23	
0x06015020	services.exe	676	632	0x06cc0080	2010-08-11 06:06:24	
0x05f47020	lsass.exe	688	632	0x06cc00a0	2010-08-11 06:06:24	
0x06384230	vmacthlp.exe	844	676	0x06cc00c0	2010-08-11 06:06:24	
0x0115b8d8	svchost.exe	856	676	0x06cc00e0	2010-08-11 06:06:24	
0x04c2b310	wscntfy.exe	888	1028	0x06cc0200	2010-08-11 06:06:49	
0x063c5560	svchost.exe	936	676	0x06cc0100	2010-08-11 06:06:24	
0x01122910	svchost.exe	1028	676	0x06cc0120	2010-08-11 06:06:24	
0x049c15f8	TPAutoConnect.exe	1084	1968	0x06cc0220	2010-08-11 06:06:52	
0x061ef558	svchost.exe	1088	676	0x06cc0140	2010-08-11 06:06:25	
0x04a544b0	ImmunityDebugger	1136	1724	0x06cc02a0	2010-08-11 16:50:19	
0x0640ac10	msiexec.exe	1144	420	0x06cc0340	2010-08-11 16:49:33	2010-08-11 16:50:08
0x06499b80	svchost.exe	1148	676	0x06cc0160	2010-08-11 06:06:26	
0x005f23a0	rundll32.exe	1260	1724	0x06cc0360	2010-08-11 16:50:29	2010-08-11 16:50:42
0x0113f648	1_doc_RCData_61	1336	1136	0x06cc0340	2010-08-11 16:50:20	
0x06945da0	spoolsv.exe	1432	676	0x06cc01a0	2010-08-11 06:06:26	
0x069d5b28	vmtoolsd.exe	1668	676	0x06cc01c0	2010-08-11 06:06:35	
0x04a065d0	explorer.exe	1724	1708	0x06cc0280	2010-08-11 06:09:29	
0x0655fc88	VMUpgradeHelper	1788	676	0x06cc01e0	2010-08-11 06:06:38	
0x0211ab28	TPAutoConnSvc.exe	1968	676	0x06cc0260	2010-08-11 06:06:39	

The listing from the *psscan* plugin appears moderately similar to the output of the *pslist* plugin. However, several processes are listed for the first time herein. Moreover, process *1_doc_RCData_61* has a very suspicious name, likely indicative of a non-Windows process, tool or application. It has been highlighted in red above. Finally, consider that malware commonly uses non-Windows names for their launched processes.

2.3.1.4 Differentiating the output between the pslist and psscan plugins

Highlighting the differences between the output from the *pslist* and *psscan* plugins may not be obvious at first glance. For this task, shell-based text processing is of significant use. By using the following command, it is readily possible to differentiate the between the two plugins' output:

```
$ cat pslist.txt psscan.txt | awk '{print $2"\t"$3}' | sort |
uniq -c | grep -v "    2"
```

This command results in the following output:

```
1 --------------------  ------
1 ----------------      ------
1 1_doc_RCData_61 1336
1 msiexec.exe     1144
1 rundll32.exe    1260
```

Thus, by using these commands, it was determined that the differences between these two plugins (*pslist* and *psscan*) are processes *1_doc_RCData_61*, *msiexec.exe* and *rundll32.exe*.

2.3.1.5 Psxview plugin

Volatility provides an additional capability for detecting hidden running processes. The *psxview* plugin provides a detailed listing of processes running in a memory image by using five specific process detection methods. These include *pslist*, *psscan*, *thrdproc*, *pspcdid* and *csrss*.

Consider the following output from this plugin, using command "*volatility -f prolaco.vmem psxview*":

Table 6: Volatility output for the Psxview plugin sorted by PID (Prolaco).

Offset(P)	Name	PID	pslist	psscan	thrdproc	pspcdid	csrss
0x01214660	System	4	True	True	True	True	False
0x05f027e0	alg.exe	216	True	True	True	True	True
0x04be97e8	VMwareTray.exe	432	True	True	True	True	True
0x04b5a980	VMwareUser.exe	452	True	True	True	True	True
0x010f7588	wuauclt.exe	468	True	True	True	True	True
0x05471020	smss.exe	544	True	True	True	True	False
0x066f0da0	csrss.exe	608	True	True	True	True	False
0x066f0978	winlogon.exe	632	True	True	True	True	True
0x06015020	services.exe	676	True	True	True	True	True
0x05f47020	lsass.exe	688	True	True	True	True	True
0x06384230	vmacthlp.exe	844	True	True	True	True	True
0x0115b8d8	svchost.exe	856	True	True	True	True	True
0x04c2b310	wscntfy.exe	888	True	True	True	True	True
0x063c5560	svchost.exe	936	True	True	True	True	True
0x01122910	svchost.exe	1028	True	True	True	True	True
0x049c15f8	TPAutoConnect.e	1084	True	True	True	True	True

Offset(P)	Name	PID	pslist	psscan	thrdproc	pspcdid	csrss
0x061ef558	svchost.exe	1088	True	True	True	True	True
0x04a544b0	ImmunityDebugge	1136	True	True	True	True	True
0x0640ac10	msiexec.exe	1144	False	True	False	False	False
0x06499b80	svchost.exe	1148	True	True	True	True	True
0x005f23a0	rundll32.exe	1260	False	True	False	False	False
0x0113f648	1_doc_RCData_61	1336	False	True	True	True	True
0x06945da0	spoolsv.exe	1432	True	True	True	True	True
0x069d5b28	vmtoolsd.exe	1668	True	True	True	True	True
0x04a065d0	explorer.exe	1724	True	True	True	True	True
0x0655fc88	VMUpgradeHelper	1788	True	True	True	True	True
0x0211ab28	TPAutoConnSvc.e	1968	True	True	True	True	True

Note that some processes listed as hidden using the *csrss* method are not always hidden. For Windows 7 and Vista systems, the list of internal processes is not available and in some cases for Windows XP required memory pages may have been swapped out, thereby affecting the outcome. [19]

However, what is not normal is that the three processes identified in the previous subsection (Section 2.3.1.4) are hidden, at a minimum, from *pslist*. These hidden processes are highlighted in red above.

2.3.1.6 Correlating PIDs and PPIDs

Examining the output established thus far based on the *pslist*, *psscan* and *psxview* plugins, the following information can be established with respect to process instantiation.

Table 7: Process instantiation for suspicious processes (Prolaco).

PPID name	PPID	PID	PID name
N/A	1708	1724	explorer.exe
explorer.exe	1724	1136	ImmunityDebugger
explorer.exe	1724	1260	rundll.exe
ImmunityDebugger	1136	1336	1_doc_RCData_61

Thus, it can be inferred that from process *explorer.exe* (PID 1724), the *Immunity Debugger* (PID 1136) was launched and from therein the Prolaco worm (PID 1336) was instantiated within the Debugger.

Based on this chain, it is unlikely that process *msiexec.exe* (PID 1144), which was previously thought of as potentially suspicious, should continue to be suspected as no information whatsoever could be found concerning its instantiating process PPID 420.

The next step is to determine if other plugins can reveal evidence of this infection.

2.3.2 Step 2: Assess other sources of evidence

This step examines various Volatility plugins that can be used to establish additional evidence concerning the memory image.

2.3.2.1 Cmdscan and consoles plugins

Plugins *cmdscan* and *consoles* may reveal additional information about commands typed into a command shell. Querying a memory image using these two plugins is carried out using the following commands:

```
$ volatility -f prolaco.vmem cmdscan

$ volatility -f prolaco.vmem consoles
```

These two plugins revealed absolutely no information whatsoever.

2.3.2.2 Connscan plugin

The first network-based Volatility plugin that should be used is *connscan*. It is used to verify the existence of ongoing network connections and scans a memory image for current or recently terminated connections.

Consider the following output from this plugin, using command "*volatility -f prolaco.vmem connscan*":

Table 8: Volatility output for the Connscan plugin (Prolaco).

Offset(P)	Local Address	Remote Address	PID
0x02214988	172.16.176.143:1148	67.208.216.86:80	1136
0x06015ab0	172.16.176.143:1149	67.208.216.86:80	1136

Based on this information, PID 1136 is the Immunity Debugger and it is in the process of communicating with an Immunity-specific computer system. More specifically, web searches reveal that computer system 67.208.216.86 is in fact computer system *debugger.immunityinc.com* and that this IP address is registered to TERRENAP DATA CENTERS, INC.

Thus, whatever was exchanged in that traffic stream, while likely relating to the Prolaco worm, was not initiated by it. Instead, it appears to have been established by the Immunity Debugger.

2.3.2.3 Connections plugin

The *connections* plugin can be used to determine information concerning recently terminated and ongoing communications. It therefore makes sense to use this plugin to query a memory image for additional network-based information.

However, using command "*volatility -f prolaco.vmem connections*" yielded no output whatsoever.

2.3.2.4 Sockets and sockscan plugins

Volatility offers two additional network-based plugins, *sockets* and *sockscan*. The *sockets* plugin lists open sockets that may provide additional information about covert network channels, while the *sockscan* plugin scans a suspect memory image for all TCP sockets. Generally, the output is the same for both plugins with the exception of memory addresses, where the *sockets* plugin uses virtual memory addressing while the *sockscan* plugin uses physical memory addressing.

Thus, using the following commands it will be possible to determine which processes have open networks sockets ready for communications:

```
$ volatility -f prolaco.vmem sockets > sockets.txt

$ volatility -f prolaco.vmem sockscan > sockscan.txt

$ cat sockets.txt sockscan.txt | sort | awk '{$1="";print}' |
uniq > sockets_sockscan.txt
```

The output of file *sockets_sockscan.txt* appears as shown in the following table:

Table 9: Volatility Sockets and Sockscan plugin output sorted by PID (Prolaco).

PID	Port	Proto	Protocol	Address	Create Time
4	445	6	TCP	0.0.0.0	2010-08-11 06:06:17
4	445	17	UDP	0.0.0.0	2010-08-11 06:06:17
4	0	47	GRE	0.0.0.0	2010-08-11 06:08:00
4	1033	6	TCP	0.0.0.0	2010-08-11 06:08:00
4	139	6	TCP	172.16.176.143	2010-08-11 06:06:28
4	137	17	UDP	172.16.176.143	2010-08-11 06:06:28
4	138	17	UDP	172.16.176.143	2010-08-11 06:06:28
216	1026	6	TCP	127.0.0.1	2010-08-11 06:06:39
688	500	17	UDP	0.0.0.0	2010-08-11 06:06:35
688	4500	17	UDP	0.0.0.0	2010-08-11 06:06:35
688	0	255	Reserved	0.0.0.0	2010-08-11 06:06:35
936	135	6	TCP	0.0.0.0	2010-08-11 06:06:24

PID	Port	Proto	Protocol	Address	Create Time
1028	123	17	UDP	127.0.0.1	2010-08-11 06:06:39
1028	123	17	UDP	172.16.176.143	2010-08-11 06:06:39
1088	1025	17	UDP	0.0.0.0	2010-08-11 06:06:38
1088	1147	17	UDP	0.0.0.0	2010-08-11 16:50:22
1148	1900	17	UDP	172.16.176.143	2010-08-11 06:06:39
1148	1900	17	UDP	127.0.0.1	2010-08-11 06:06:39

Looking at this data, based on the various PIDs and PPIDs of interest (1136 (*Immunity Debugger*), 1144 (*msiexec.exe*), 1260 (*rundll32.exe*), 1336 (*1_doc_RCData_1*), 1708 (unknown) and 1724 (*explorer.exe*)) established thus far, none of the port-enabled processes listed in this table corresponds to these processes.

Thus, when the results of the *sockets* and *sockscan* plugins are taken together with the results of the *connscan* and *connections* plugins, it appears that the Prolaco worm is not currently in the midst of initiating or maintaining any network connections, sockets or streams.

2.3.2.5 Filescan plugin

If an infection is active and does not show itself via the network then the *filescan* plugin may be of assistance as the plugin may be able to find open file handles in memory. Unfortunately, no direct link to these handles is possible as the physical disk image is not available for analysis. This plugin makes use of physical address offsets.

The preferred method for detecting indicators of compromise is twofold. First, using keywords (e.g. Prolaco, infection, rootkit, worm, etc.) it may be possible to find the infection, as malware programmers do not often use innocuous looking filenames. Of course, this is at best a hit and miss approach. Secondly, it can be attempted to detect suspicious files based on their locations. However, this requires that the investigator has a very good working knowledge of the underlying operating system as just looking at filenames[1] and locations will not produce meaningful results, unless something really sticks out.

However, unlike the Zeus memory analysis report [22], there is little useful information available on the web concerning the Prolaco worm, thus, few indicators can be readily used as keywords. Using additional filename keyword information as provided in [20], additional keyword filters could be used in conjunction with the *filescan* plugin.

Running command "*volatility -f prolaco.vmem filescan | grep -i -P '(1_doc_RCData_612|virus|Trojan|rootkit|worm|Prolaco|rundll|msiexec|google|wmimngr|jusched|wfmngr|wupmgr|java|wpmgr|nvscpapisvr)'* " results in the following output:

[1] Recall that a reliable source of filenames is the NSRL hash-set. It can be broken down by software product and operating system.

```
0x010bd028                          1                          0    R--r-d
\Device\HarddiskVolume1\WINDOWS\system32\rundll32.exe

0x01188cb8      1      0 R--rw- \Device\HarddiskVolume1\Documents
and Settings\Administrator\Desktop\1_doc_RCData_612.exe

0x04a97248                          1                          0    R--r-d
\Device\HarddiskVolume1\WINDOWS\system32\msiexec.exe

0x05fbed48                          1                          0    R--r-d
\Device\HarddiskVolume1\WINDOWS\system32\rundll32.exe

0x07fbc538      1      0 R--r-d \Device\HarddiskVolume1\Documents
and Settings\Administrator\Desktop\1_doc_RCData_612.exe
```

Based on this output, it is readily established that processes *msiexec.exe* and *rundll32.exe* are located in *C:\Windows\system32*. Thus, there is little reason to continue suspecting these two processes anymore. However, the malware's suspicious process has been clearly (*1_doc_RCData_612.exe*) found and it is highlighted in red in the above output.

2.3.2.6 Mutantscan plugin

The Volatility *mutantscan* can sometimes reveal interesting information about Windows thread-based mutexes in memory. It makes use of physical offset addressing.

Using command "*volatility -f prolaco.vmem mutantscan*" yielded the following pertinent information after pruning the output (as the output is several pages long):

```
0x010ac258      2       1          0  0xff144c08  1336:1204
GoogleUpte.exeDm28sf0V@XK$NX8hOu
```

This output indicates that suspicious process PID 1336 (*1_doc_RCData_61*) is relying on a mutex named "*GoogleUpte.exeDm28sf0V@XK$NX8hOu*." It can be inferred that this highly suspicious process is making use of some potentially malicious mutex with an innocuous looking name so as not to cause alarm.

2.3.2.7 Handles plugin

The Volatility *handles* plugin can reveal interesting information about processes and resources attached or associated to them that might not be found using previously examined plugins. It makes use of virtual offset addressing.

Using command "*volatility -f prolaco.vmem handles*," the following pruned output is of interest to the investigation and is as follows:

```
0x80fdc648  608   0x2b4   0x1f0fff   Process  1_doc_RCData_61(1336)

0xff144c08  608   0x3c8   0x1f03ff   Thread   TID 1204 PID 1336

0x80fdc648  1136  0x124   0x12067b   Process  1_doc_RCData_61(1336)
```

```
0x80fdc648  1136  0x78   0x1f0fff  Process  1_doc_RCData_61(1336)
0xff144c08  1136  0x11c  0x12007b  Thread   TID 1204 PID 1336
0xff144c08  1136  0x7c   0x1f03ff  Thread   TID 1204 PID 1336
```

From this output, it can be determined that thread TID 1204 is a thread of process PID 1336 (*1_doc_RCData_61*), highlighted in red above. Moreover, PID 1336 very likely relies on, at least partially, certain Windows-based console functions provided by *csrss.exe* (PID 608) [21], highlighted in yellow.

2.3.2.8 Threads plugin

The final Volatility plugin to be used in this step is the *threads* plugin. Armed with the information provided by the *handles* plugin, it is worthwhile investigating the information uncovered about TID 1204. Using command "*volatility -f prolaco.vmem threads -p 1336*" yields the following information:

```
ETHREAD: 0x00353c08 Pid: 1336 Tid: 1204
Tags: ScannerOnly
Created: 2010-08-11 16:50:20
Exited: 1970-01-01 00:00:00
Owning Process: 1_doc_RCData_61
Attached Process: 1_doc_RCData_61
State: Waiting:Executive
BasePriority: 0x8
Priority: 0xa
TEB: 0x7ffdf000
StartAddress: 0x7c810867 UNKNOWN
ServiceTable: 0x80552140
  [0] 0x80501030
  [1] 0xbf997600
  [2] 0x00000000
  [3] 0x00000000
Win32Thread: 0xe1702858
CrossThreadFlags:
```

Thus, suspicious thread TID 1204 is without doubt a subset (i.e. thread) of PID 1336.

2.3.3 Step 3: Dump and assess suspicious process

The evidence established thus far indicates that one process, PID 1336 (*1_doc_RCData_61*) is very likely the infection sought after. This step examines various methods for dumping the process and then evaluating it.

2.3.3.1 Create data directories

Create directories *malfind*, *memdump*, *procexedump* and *procmemdump* for storing memory samples that are to be dumped from the memory image using corresponding Volatility plugins. This is done using the following commands:

```
$ mkdir malfind

$ mkdir memdump

$ mkdir procexedump

$ mkdir procmemdump
```

2.3.3.2 Malfind plugin

Volatility's *malfind* plugin was specifically designed to search for malware hidden through code injection. If memory address offsets are specified then they must be physical memory address offsets.

Using command "*volatility -f prolaco.vmem -p 1336 -o 0x0113f648 malfind --dump-dir=malfind*" it was attempted to find and dump injected malicious code associated with process *1_doc_RCData_612.exe*. This command resulted in no output.

Attempting to use this plugin at large against the infected memory image, using command "*volatility -f prolaco.vmem malfind*" resulted in the following non-pertinent output:

```
Process: csrss.exe Pid: 608 Address: 0x7f6f0000
Vad Tag: Vad  Protection: PAGE_EXECUTE_READWRITE
Flags: Protection: 6

0x7f6f0000  c8 00 00 00 ff 01 00 00 ff ee ff ee 08 70 00 00   .............p..
0x7f6f0010  08 00 00 00 00 fe 00 00 00 00 10 00 00 20 00 00   ................
0x7f6f0020  00 02 00 00 00 20 00 00 8d 01 00 00 ff ef fd 7f   ................
0x7f6f0030  03 00 08 06 00 00 00 00 00 00 00 00 00 00 00 00   ................

0x7f6f0000 c8000000          ENTER 0x0, 0x0
0x7f6f0004 ff01              INC DWORD [ECX]
0x7f6f0006 0000              ADD [EAX], AL
0x7f6f0008 ff                DB 0xff
0x7f6f0009 ee                OUT DX, AL
0x7f6f000a ff                DB 0xff
0x7f6f000b ee                OUT DX, AL
0x7f6f000c 087000            OR [EAX+0x0], DH
0x7f6f000f 0008              ADD [EAX], CL
0x7f6f0011 0000              ADD [EAX], AL
0x7f6f0013 0000              ADD [EAX], AL
0x7f6f0015 fe00              INC BYTE [EAX]
0x7f6f0017 0000              ADD [EAX], AL
0x7f6f0019 0010              ADD [EAX], DL
0x7f6f001b 0000              ADD [EAX], AL
0x7f6f001d 2000              AND [EAX], AL
0x7f6f001f 0000              ADD [EAX], AL
0x7f6f0021 0200              ADD AL, [EAX]
0x7f6f0023 0000              ADD [EAX], AL
0x7f6f0025 2000              AND [EAX], AL
0x7f6f0027 008d010000ff      ADD [EBP-0xffffff], CL
0x7f6f002d ef                OUT DX, EAX
0x7f6f002e fd                STD
0x7f6f002f 7f03              JG 0x7f6f0034
0x7f6f0031 0008              ADD [EAX], CL
0x7f6f0033 06                PUSH ES
0x7f6f0034 0000              ADD [EAX], AL
```

```
0x7f6f0036 0000              ADD [EAX], AL
0x7f6f0038 0000              ADD [EAX], AL
0x7f6f003a 0000              ADD [EAX], AL
0x7f6f003c 0000              ADD [EAX], AL
0x7f6f003e 0000              ADD [EAX], AL
```

Thus, based on this plugin's output, no malicious injected code was found.

Because no code injection was found for this memory image, there is no point in subsequent plugins (memdump, procexedump, procmemdump) in attempting to perform additional analyses of processes other than PID 1336 (*1_doc_RCData_61*) .

2.3.3.3 Memdump plugin

The *memdump* plugin is used to dump the addressable memory space of a given process. If memory address offsets are specified then they must be physical memory address offsets. The plugin dumps all data segments associated with the specified process to a single destination file. The plugin is worth trying as the information attained from it, combined with any potential information obtained from the *malfind*, *procexedump* and *procmemdump* plugins could be used in the reverse engineering of the malware. The process likely to bear fruit from this memory image, based on the analyses conducted thus far, is PID 1336.

The command used to dump the addressable memory space of PID 1336 was:

```
$ volatility -f prolaco.vmem memdump -p 1336 -o 0x0113f648 --
dump-dir=memdump
```

The file dumped as a result of this command had the following metadata:

Table 10: Metadata for PID 1336 dumped using the Memdump plugin (Prolaco).

Filename	memdump/1336.dmp
Size	59,535,360 bytes
SHA1 hash	587bb4c246c42df4f8b10f7037807f2546d10b04
Fuzzy hash	393216:h2GN5vGNE+q329LVCH2BQNLCi/Jfac7nEAsibk6kpPyL:h2Gjd+S29Lm2BQNLVacgA3bk6k5y

All file metadata information was saved for potential future use.

2.3.3.4 Procexedump plugin

Unlike the *memdump* plugin, the *procexedump* plugin dumps only a process' executable code. If memory address offsets are specified then they must be physical memory address offsets.

The process likely to bear fruit from this memory image, based on the analyses conducted thus far, is PID 1336. The command used to this process' executable code was:

```
$ volatility -f prolaco.vmem procexedump -p 1336 -o 0x0113f648 --
dump-dir=procexedump
```

The file dumped as a result of this command had the following metadata:

Table 11: Metadata for PID 1336 dumped using the Procexedump plugin (Prolaco).

Filename	procexedump/executable.1336.exe
Size	318,976 bytes
SHA1 hash	275f5321a563aa3e981001d94f8e863e2e22da3f
Fuzzy hash	6144:ZUg06lHuzJ2MZ7zB4H2Qy43Wk2LAyrRlEFcA9MQvM:ZUg0SOzYMZ7zB4H2i3WpU79jU

All file metadata was saved for potential future use.

2.3.3.5 Procmemdump plugin

Unlike the *memdump* and *procexedump* plugins, the *procmemdump* plugin dumps a process' executable code including associated slack space (all processes have some slack space). If memory address offsets are specified then they must be physical memory address offsets.

The process likely to bear fruit from this memory image, based on the analyses conducted thus far, is PID 1336. The command used to dump the process' executable code and slack space was:

```
$ volatility -f prolaco.vmem procmemdump -p 1336 -o 0x0113f648 --
dump-dir=procmemdump
```

The file dumped as a result of these commands had the following metadata:

Table 12: Metadata for PID 1336 dumped using the Procmemdump plugin (Prolaco).

Filename	procmemdump/executable.1336.exe
Size	348,160 bytes
SHA1 hash	bbfa3a5dcbcff29e600c863e8c722c738bab9082
Fuzzy hash	6144:sUg06lHuzJ2MZ7zB4H2/y43Wk2LAyrRlEFcA9MQvM:sUg0SOzYMZ7zB4H2L3WpU79jU

All file metadata was saved for potential future use.

2.3.3.6 Virus scanning and hash verification of dumped processes

2.3.3.6.1 Scanner examination

Using the six anti-virus scanners against the three extracted files produced using *memdump* (*1336.dmp*), *procexedump* (*executable.1336.exe*) and *procmemdump* (*executable.1336.exe*), the following scanner results were obtained:

Table 13: Detection of infection for dumped memory files using Memdump, Procexedump and Procmemdump plugins (Prolaco)

Scanner	Plugin Output File (directory and filename)	Infection Identification
Avast	memdump/1336.dmp	Detected as non-infected
	procexedump/executable.1336.exe	Infected by Win32:Malware-gen
	procmemdump/executable.1336.exe	Infected by Win32:Malware-gen
AVG	memdump/1336.dmp	Detected as non-infected
	procexedump/executable.1336.exe	Infected by Worm/Generic2.FQ
	procmemdump/executable.1336.exe	Infected by Worm/Generic2.FQ
BitDefender	memdump/1336.dmp	Detected as non-infected
	procexedump/executable.1336.exe	Infected by Gen:Trojan.Heur.tyW@Xw!!9Rai
	procmemdump/executable.1336.exe	Infected by Gen:Trojan.Heur.vyW@XUoM8Moi
ClamAV	memdump/1336.dmp	Detected as non-infected
	procexedump/1336.exe	Detected as non-infected
	procmemdump/1336.exe	Detected as non-infected
FRISK	memdump/1336.dmp	Detected as non-infected
	procexedump/executable.1336.exe	Detected as encrypted ZIP file
	procmemdump/executable.1336.exe	Detected as encrypted ZIP file
McAfee	memdump/1336.dmp	Detected as non-infected
	procexedump/executable.1336.exe	Detected as non-infected
	procmemdump/executable.1336.exe	Infected by Generic.dx!7B0F17C2C2E3

Only dumped file *procexedump/executable.1336.dmp* (obtained from plugin *procexedump*) was found to match one of the scanner-specific infection detections. Specifically, this dumped file received the same scanner message from the AVG scanner when it scanned carved data file *recup_dir.2/f0139704.exe*. Both files received the detection of **Worm/Generic2.FQ**.

File *memdump/1336.dmp* was detected as uninfected by all scanners. Only files *procexedump/executable.1336.exe* and *procmemdump/executable.1336.exe* were detected as infected by some of the scanners. More specifically, the former was detected by three scanners while the latter was detected by four scanners.

2.3.3.6.2 Hash comparison

When comparing the fuzzy hashes of the three extracted files for PID 1336 (*memdump*, *procexedump* and *procmemdump*) against the carved data files no matches were found. Thus, what was recovered through data carving and what was recovered through Volatility differ so greatly that even fuzzy hash matching determined that no discernible similarities could be found between them.

However, between dumped files *procexedump/executable.1336.exe* and *procmemdump/executable.1336.exe*, it was found that they had a 96% fuzzy hash similarity, indicating they were very similar to one another. No fuzzy hash matches could be established between *procexedump/executable.1336.exe* and *memdump/1336.dmp* or between *procmemdump/executable.1336.exe* and *memdump/1336.dmp*.

SHA1 hash matching of the three Volatility recovered files yielded no matches against either the NSRL or the carved data files.

2.3.4 Step 4: Examining the registry

The Windows registry serves to complicate and facilitate the investigator's work. It is commonly used by malware to configure system settings for permanent infection. However, the difficulty in working with the registry lies in knowing where to look. The registry is spread out across many data files (also commonly known as registry hives) in various locations and each serves a specific purpose with respect to system, application and user configurations. Annex D provides a listing of registry keys commonly used by malware.

2.3.4.1 Hivelist plugin

The purpose of using the *hivelist* plugin is to determine which registry hives[2] are available in the memory image.

Consider the following output from this plugin, using command "*volatility -f prolaco.vmem hivelist*":

[2] A registry hive denotes the actual disk file and its location on disk.

Table 14: Volatility output for the Hivelist plugin (Prolaco)

Virtual Address	Physical Address	Filename and Location
0xe1c49008	0x036dc008	\Device\HarddiskVolume1\Documents and Settings\LocalService\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
0xe1c41b60	0x04010b60	\Device\HarddiskVolume1\Documents and Settings\LocalService\NTUSER.DAT
0xe1a39638	0x021eb638	\Device\HarddiskVolume1\Documents and Settings\NetworkService\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
0xe1a33008	0x01f98008	\Device\HarddiskVolume1\Documents and Settings\NetworkService\NTUSER.DAT
0xe153ab60	0x06b7db60	\Device\HarddiskVolume1\WINDOWS\system32\config\software
0xe1542008	0x06c48008	\Device\HarddiskVolume1\WINDOWS\system32\config\default
0xe1537b60	0x06ae4b60	\Device\HarddiskVolume1\WINDOWS\system32\config\SECURITY
0xe1544008	0x06c4b008	\Device\HarddiskVolume1\WINDOWS\system32\config\SAM
0xe13ae580	0x01bbd580	[no name]
0xe101b008	0x01867008	\Device\HarddiskVolume1\WINDOWS\system32\config\system
0xe1008978	0x01824978	[no name]
0x8066e904	0x0066e904	[no name]
0xe1e158c0	0x009728c0	\Device\HarddiskVolume1\Documents and Settings\Administrator\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
0xe1da4008	0x00f6e008	\Device\HarddiskVolume1\Documents and Settings\Administrator\NTUSER.DAT

2.3.4.2 Printkey plugin

Using the proposed registry keys identified in Annex D, 952 Volatility *printkey* commands were issued via a script to query the memory image for information pertaining to the various registry hives where this malware may have left traces of its activity. All output was captured and stored in a text file for further analysis.

After running the script, no pertinent information concerning the infection could be found.

2.3.4.3 Userassist plugin

The final registry-based Volatility plugin run against the memory image was *userassist*. This plugin has the potential to provide, among other things, registry-based information pertaining to programs run and files opened by the user.

Unfortunately, this plugin did not result in any useful information concerning the infection.

2.3.5 Step 5: Strings analysis

Another technique must be used to extract pertinent information from the memory image concerning the infection. Thus, using the *strings* command it may be possible to obtain additional evidence about the malware and its effect on the underlying computer system.

2.3.5.1 Extraction against plugin-based dumped files

This subsection conducts *strings*-based analysis against only those files successfully obtained using the memory dumping plugins (*procexedump* and *procmemdump*).

2.3.5.1.1 Commands

The following commands were used against the *procexedump* and *procmemdump* plugin-dumped files for PID 1336:

```
$  strings -e s -t d procexedump/executable.1336.exe | grep -i -P
‘(keyword1|keyword2|…|keywordn)’

$  strings -e S -t d procexedump/executable.1336.exe | grep -i -P
‘(keyword1|keyword2|…|keywordn)’

$  strings -e l -t d procexedump/executable.1336.exe | grep -i -P
‘(keyword1|keyword2|…|keywordn)’

$  strings -e L -t d procexedump/executable.1336.exe | grep -i -P
‘(keyword1|keyword2|…|keywordn)’

$  strings -e s -t d procmemdump/executable.1336.exe | grep -i -P
‘(keyword1|keyword2|…|keywordn)’

$  strings -e S -t d procmemdump/executable.1336.exe | grep -i -P
‘(keyword1|keyword2|…|keywordn)’

$  strings -e l -t d procmemdump/executable.1336.exe | grep -i -P
‘(keyword1|keyword2|…|keywordn)’

$  strings -e L -t d procmemdump/executable.1336.exe | grep -i -P
‘(keyword1|keyword2|…|keywordn)’
```

These commands carryout case-insensitive (-i) searches using *grep*’s Perl-like (-P) pattern matching; to remove non-pertinent output keyword filters are used. Output can be tuned where *keyword1*, *keyword2*… *keywordn* represent the following word-filters:

Filter Set (1):

- `1_doc_RCData_612`
- `Virus`
- `Trojan`

- Rootkit
- Worm
- Prolaco
- Shield
- Infected
- Software\\
- Software\\Microsoft
- System\\
- System\\CurrentControlSet
- System\\ControlSet

Consider that word-filters *Software*\\, *System*\\ and *CurrentVersion*\\ (see Filter Set (2) below) are indicative of registry hives.

2.3.5.1.2 Pertinent strings

Running the aforementioned commands with the above listed keyword filters resulted in the following pertinent strings, likely applicable to this specific malware:

- SOFTWARE\McAfee\AVEngine
- mcshield
- Software\Microsoft\Windows\CurrentVersion\Run
- SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System
- SOFTWARE\Microsoft\Security Center

2.3.5.1.3 Analysis

Based on the above output, the malware may be attempting to seek out specific anti-virus products. Thus, the word-filter can be expanded to incorporate additional filters to search for including:

Filter Set (2):

- Mcafee
- mcshield
- CurrentVersion\\Run
- CurrentVersion\\Policies\\System
- Security Center

2.3.5.2 Extraction against memory image

This subsection conducts *strings*-based analysis against the entire memory image file, *prolaco.vmem*.

2.3.5.2.1 Commands

Using the aforementioned keywords (see Filter Set (1) and (2)), the following commands were run against the memory image:

```
$ strings -e s -t d prolaco.vmem | grep -i -P
‘(keyword1|keyword2|…|keywordn)’

$ strings -e S -t d prolaco.vmem | grep -i -P
‘(keyword1|keyword2|…|keywordn)’

$ strings -e l -t d prolaco.vmem | grep -i -P
‘(keyword1|keyword2|…|keywordn)’

$ strings -e L -t d prolaco.vmem | grep -i -P
‘(keyword1|keyword2|…|keywordn)’
```

2.3.5.2.2 Pertinent strings

Applying Filter Set (1) and (2) to the aforementioned *strings* commands, the following output has been manually pruned for pertinence. Thus, it is possible that some items will have been inadvertently missed. The pertinent output is as follows:

- 1_doc_RCC:\Documents and Settings\Administrator\Desktop\1_doc_RCData_612.exe
- 1_doc_RCData_612
- \1_doc_RCData_612.exe
- 1_doc_RCData_612.exe
- 1_doc_RCData_612.exe|
- 1_DOC_RCDATA_612.EXE
- 1_doc_RCData_612.exe1_DOC_RCDATA_612.EXEI
- 1_DOC_RCDATA_612.EXE-2A36E0B8.pf
- 1_DOC_RCDATA_612.EXE-2A36E0B8.pf1_DOC_RCDATA_612.EXE-2A36E0B8.PF
- 1_doc_RCData_612.exe3_s
- 1_DOC_RCDATA_612.EXEo
- ALS~1\Temp\VMwareDnD\bffef9ba\1_doc_RCData_612.exe
- bvShieldEnabled
- C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\VMwareDnD\bffef9ba\1_doc_RCData_612.exe
- "C:\Documents and Settings\Administrator\Desktop\1_doc_RCData_612.exe
- "C:\Documents and Settings\Administrator\Desktop\1_doc_RCData_612.exe"

- "C:\Documents and Settings\Administrator\Desktop\1_doc_RCData_612.exe"‘|€¦]
- "C:\Documents and Settings\Administrator\Desktop\1_doc_RCData_612.exe"‘|€¦]
-]C:\Documents and Settings\Administrator\Desktop\1_doc_RCData_612.exe
- \??\C:\Documents and Settings\Administrator\Desktop\1_doc_RCData_612.exe
- C:\Documents and Settings\Administrator\Desktop\1_doc_RCData_612.exe
- ??\C:\Documents and Settings\Administrator\Desktop\1_doc_RCData_612.exe.Config
- ??\C:\Documents and Settings\Administrator\Desktop\1_doc_RCData_612.exe.Manifest
- Command line: "C:\Documents and Settings\Administrator\Desktop\1_doc_RCData_612.exe"T
- "C:\Program Files\Immunity Inc\Immunity Debugger\ImmunityDebugger.exe" "C:\Documents and Settings\Administrator\Desktop\1_doc_RCData_612.exe"
- \Device\HarddiskVolume1\Documents and Settings\Administrator\Desktop\1_doc_RCData_612.exe
- \DEVICE\HARDDISKVOLUME1\DOCUMENTS AND SETTINGS\ADMINISTRATOR\DESKTOP\1_DOC_RCDATA_612.EXE
- DEVICE\HARDDISKVOLUME1\DOCUMENTS AND SETTINGS\ADMINISTRATOR\DESKTOP\1_DOC_RCDATA_612.EXE
- \DOCUME~1\ADMINI~1\LOCALS~1\Temp\VMwareDnD\bffef9ba\1_doc_RCData_612.exe
- ÿÿÿÿ\DOCUME~1\ADMINI~1\LOCALS~1\Temp\VMwareDnD\bffef9ba\1_doc_RCData_612.exe
- `\Documents and Settings\Administrator\Desktop\1_doc_RCData_612.exe
- \Documents and Settings\Administrator\Desktop\1_doc_RCData_612.exe
- Documents and Settings\Administrator\Desktop\1_doc_RCData_612.exe"
- :\Documents and Settings\Administrator\Desktop\1_doc_RCData_612.exe.Config
- \Documents and Settings\Administrator\Desktop\1_doc_RCData_612.exe.Manifest
- File 'C:\Documents and Settings\Administrator\Desktop\1_doc_RCData_612.exe'K/
- Immunity Debugger - 1_doc_RCData_612.exe

- Immunity Debugger - 1_doc_RCData_612.exe - [CPU - main thread, module 1_doc_RC]
- McAfeeAntiVirus
- McAfeeFirewall
- McAfee Firewall
- McAfee Quick Clean 1.02
- McAfee Software
- McAfee Uninstaller 3.0
- mcshield
- mcshield.exe
- Modules C:\Documents and Settings\Administrator\Desktop\1_doc_RCData_612.exeL'
- OCUME~1\ADMINI~1\LOCALS~1\Temp\VMwareDnD\bffef9ba\1_doc_RCData_612.exe
- ocuments and Settings\Administrator\Desktop\1_doc_RCData_612.exe
- Rundll32.exe C:\PROGRA~1\COMMON~1\System\OLEDB~1\oledb32.dll,OpenDSLFile %1
- RY\MACHINE\SYSTEM\CURRENTCONTROLSET\SERVICES\TSDDD\DEVICE0
- s/ÏÂ'ôõ÷ôõ÷òóõðóôïòóïòóïòóïòóïòóïòóïALS~1\Temp\VMwareDnD\bffef9ba\1_doc_RCData_612.exe
- SHIELD.BPL
- Shielded
- SOFTWARE\McAfee\AVEngine
- Software\McAfee.com\Agent\Apps
- Software\McAfee.com\Agent\Apps\MPF
- Software\McAfee.com\Agent\Apps\VSO
- Software\McAfee.com\Personal Firewall
- Software\McAfee.com\VirusScan Online
- Software\McAfee\McAfee Firewall
- tes\Anti-Virus\Resident
- trator\Desktop\1_doc_RCData_612.exe
- Unshielded
- /Users/mhl/Desktop/MHLFILES/iDefense/Googlebuzz/1_doc_RCData_612.exe
- VIRUS: kernel32.exe
- VirusProduct
- VirusScanner

2.3.5.2.3 Analysis

Examining the above output, the overall impression is that this string listing requires additional context to make sense of certain information. However, what can be inferred is that the malware appears to perform checks for McAfee-based software security products. Whether the malware was actually able to disable them will require reverse engineering, an approach not examined herein. Based on information provided by [5], it is known that registry entry *bvShieldEnabled* (found in the above output) corresponds to known worm behaviour with respect to McAfee Anti-Virus.

Moreover, based on the above output, the malware infection likely relied on configuration and manifest files, as per the following artifacts obtained from the above *strings* output (see Section 2.3.5.2.2 for details):

- `??\C:\Documents and Settings\Administrator\Desktop\1_doc_RCData_612.exe.Config`
- `??\C:\Documents and Settings\Administrator\Desktop\1_doc_RCData_612.exe.Manifest`

The malware was most likely introduced to the virtual machine through the VMware host-to-guest mechanism as per the above-listed *strings* artifact (see Section 2.3.5.2.2 for details):

```
\DOCUME~1\ADMINI~1\LOCALS~1\Temp\VMwareDnD\bffef9ba\1_doc_RCData_
612.exe
```

Finally, the malware appears to have been instantiated using Immunity Debugger the above-listed *strings* artifact (see Section 2.3.5.2.2 for details):

```
"C:\Program Files\Immunity Inc\Immunity
Debugger\ImmunityDebugger.exe" "C:\Documents and
Settings\Administrator\Desktop\1_doc_RCData_612.exe"
```

2.4 Summary

Although little useful information could be found in the publicly available literature concerning the Prolaco worm thought to have infected the memory image, through a systematic application of the proposed and amended methodology (see Section 1.7 for details), it was possible to not only find and extract the malware but determine that it likely had anti-scanner capabilities. Because this malware was hidden and not communicating with any remote systems, its detection and extraction was less straightforward than for the Zeus memory image (see [22] for details).

Moreover, based on the evidence obtained from the application of the aforementioned Volatility plugins against this memory image, it can be posited that the malware did not complete its infection of the underlying computer system because it was run from a debugger and had likely been paused or stopped for further analysis when the memory image was taken.

Based on the output obtained using the *mutantscan* and *handles* plugins, the worm makes no effort to hide itself.

3 Memory investigation and analysis of SpyEye

3.1 Background

This analysis examines a memory image suspected of harbouring the SpyEye Trojan horse as based on the methodology put forward in Section 1.7. Much information was found concerning the technical details of this infection as it is somewhat similar in its scope to the Zeus Trojan horse. Documents [6][7][8][9][10][11][12][13][14][15][16][17] and [18] provide a wealth of additional information for analysts of varying skill.

3.2 Preliminary investigative steps

The steps examined in this subsection should be considered as preliminary investigative steps necessary for examining a potentially infected memory image.

3.2.1 Safeguard the memory image

The memory image *spyeye.vmem* was set to immutable atop an Ext4-based filesystem. The command used to perform this, carried out as the root user, was:

```
$ sudo chattr +i spyeye.vmem
```

This results in a memory image that can no longer be modified, even by the root user. This is to prevent accidental modifications from occurring to this file.

3.2.2 Preliminary anti-virus scanning results

Scanning only the memory image itself with the six scanners outlined in Section 1.6.1, it was determined that unlike the Zeus infected memory image (see [22] for details) none of them identified the memory image as infected.

Even though no infection was found, all scanner results were saved for possible future use.

3.2.3 Data carving and file hashing

Photorec succeeded in recovering 1,495 files carved from the SpyEye memory image as per the recommended Photorec settings put forward in Section 1.6.2. Eight duplicate files were found, thereby leaving 1,487 unique files recovered. Of those 1,495 recovered files, 929 were identified as PE-based files. Of those, 678 were identified as Windows 32-bit DLLs, while 251 were identified as standard Windows 32-bit PEs and device drivers. Finally, of the Windows 32-bit PE-based files, nine were detected as UPX-based executables.

No 64-bit PE-based files were identified. However, two files were identified as 16-bit MS-DOS executables for Windows 3.x. This discovery was likely caused by incorrect header detection due to the imprecise nature of data carving.

Other file types were detected but were of no immediate use. However, their types were recorded and saved for possible future use within this analysis.

All recovered files were SHA1-hashed and then validated against NSRL hash-set 2.40 (March 2013). Results were stored for future use. Six SHA1 hashes were confirmed as matching the NSRL hash-set. Information concerning these matches can be found in Annex C.2.

Finally, CTPH-based hashing (fuzzy hashing) was conducted using the *ssdeep* tool against the carved data files and stored for future use.

3.2.4 Anti-virus scanning and file hashing results for carved data files

Using the six scanners and combining their output through UNIX command line processing tools (e.g. *cat*, *sort*, *find*, *tr*, *strings*, *awk*, *grep*, *uniq*, etc.), five matches were established. A match occurs when two or more scanners detect a file as infected or possibly malicious. Of the six scanners, only Avast was incapable of detecting any of the files as potentially malicious.

All six scanners were capable of detecting one or more potential infections. However, none of the potentially infected files were detected as the SpyEye Trojan horse or anything else remotely resembling the name of this infection. The following table provides a detailed correspondence of the scanner-based matches:

Table 15: Matching of potentially infected carved data file vs. scanner (SpyEye).

Potentially infected file	Detecting scanner
f0263584.exe	AVG
	ClamAV
f0263296.dll	AVG
	ClamAV
f0305128.dll	AVG
	ClamAV
f0630512.exe	BitDefender
	ClamAV
f0952760.dll	AVG
	ClamAV

Specific logs for each scanner can be found in Annex A.2.

3.3 Volatility analysis

In order to investigate this specific memory image the use and output of various Volatility plugins of assistance to this particular analysis are examined.

3.3.1 Step 1: Determine background information

This step examines the Volatility plugins used to provide background information and context to the memory image.

3.3.1.1 Imageinfo plugin

This Volatility plugin is used to provide basic contextual information about a suspect memory image.

Output from the plugin, using command "*volatility imageinfo -f spyeye.vmem*," is as follows:

```
Determining Determining profile based on KDBG search...

          Suggested   Profile(s)   :   WinXPSP2x86,   WinXPSP3x86
(Instantiated with WinXPSP2x86)
                     AS Layer1 : JKIA32PagedMemoryPae (Kernel AS)
                     AS       Layer2       :       FileAddressSpace
(/home/richard/work/Work
Reports/Studies/Volatility_Win_Mem_Analysis/Report2_Spyeye/spyeye
.vmem)
                       PAE type : PAE
                            DTB : 0x319000L
                           KDBG : 0x80545b60
          Number of Processors : 1
     Image Type (Service Pack) : 3
                KPCR for CPU 0 : 0xffdff000
             KUSER_SHARED_DATA : 0xffdf0000
           Image date and time : 2011-01-06 14:50:19 UTC+0000
     Image local date and time : 2011-01-06 09:50:19 -0500
```

This memory image appears to be running atop a 32-bit Windows XP computer system with Service Pack 3. It is equipped with one PAE-based processor and the memory image is 512 MiB in size (based on the memory image's size determined using *ls -l*). The memory image was acquired January 6, 2011 at 09:50:19 EST.

3.3.1.2 Pslist plugin

The next step is to determine which processes are running within the memory image in order to determine if anything suspicious is immediately visible. The *pslist* plugin provides a detailed process listing and makes use of virtual memory addresses.

Consider the following output from this plugin, using command "*volatility -f spyeye.vmem pslist*":

Table 16: Volatility output for the Pslist plugin sorted by PID (SpyEye).

Offset(V)	Name	PID	PPID	Thds	Hnds	Sess	Wow64	Start	Exit
0x825c8830	System	4	0	58	387	------	0		
0x8236d7a0	wuauclt.exe	536	1068	3	107	0	0	2010-11-11 22:03:33	
0x823fe020	smss.exe	572	4	3	19	------	0	2010-11-11 22:02:08	
0x82503220	csrss.exe	636	572	13	399	0	0	2010-11-11 22:02:13	
0x81f4c550	winlogon.exe	660	572	21	596	0	0	2010-11-11 22:02:14	
0x8207d5f0	services.exe	704	660	17	285	0	0	2010-11-11 22:02:15	
0x824264c0	lsass.exe	716	660	20	356	0	0	2010-11-11 22:02:15	
0x8230c5f8	vmacthlp.exe	872	704	2	26	0	0	2010-11-11 22:02:16	
0x8226cda0	svchost.exe	904	704	16	191	0	0	2010-11-11 22:02:16	
0x823f2020	svchost.exe	972	704	9	264	0	0	2010-11-11 22:02:17	
0x823e32f8	explorer.exe	1008	680	15	468	0	0	2010-11-11 22:02:55	
0x824578b0	imapi.exe	1040	704	5	114	0	0	2010-11-11 22:03:54	
0x822a0758	svchost.exe	1068	704	58	1256	0	0	2010-11-11 22:02:17	
0x81f4b020	svchost.exe	1108	704	7	82	0	0	2010-11-11 22:02:17	
0x82406da0	svchost.exe	1232	704	13	169	0	0	2010-11-11 22:02:18	
0x81ec2020	TSVNCache.exe	1252	1008	9	58	0	0	2010-11-11 22:02:58	
0x82436a48	svchost.exe	1456	704	12	121	0	0	2010-11-11 22:02:19	
0x81ebd300	VMwareTray.exe	1484	1008	2	51	0	0	2010-11-11 22:03:00	
0x82067858	svchost.exe	1540	704	6	95	0	0	2010-11-11 22:02:26	
0x82159958	VMwareUser.exe	1588	1008	7	230	0	0	2010-11-11 22:03:00	
0x82072660	jqs.exe	1612	704	6	149	0	0	2010-11-11 22:02:27	
0x8214ba18	jusched.exe	1672	1008	2	97	0	0	2010-11-11 22:03:00	
0x82284b80	vmtoolsd.exe	1816	704	6	268	0	0	2010-11-11 22:02:30	
0x822e69f8	VMUpgradeHelper	1872	704	4	100	0	0	2010-11-11 22:02:30	
0x82458020	alg.exe	2108	704	7	107	0	0	2010-11-11 22:03:54	
0x82226b48	cleansweep.exe	2268	1008	0	-------	0	0	2011-01-06 14:36:52	2011-01-06 14:36:52
0x820bd760	gmer.exe	2728	1008	2	33	0	0	2011-01-06 14:37:41	
0x82389020	wscntfy.exe	2772	1068	2	29	0	0	2010-11-11 22:03:56	
0x81f7a708	WPFFontCache_v0	3084	704	7	70	0	0	2010-11-11 22:05:04	
0x81f5e020	jucheck.exe	3892	1672	3	105	0	0	2010-11-11 22:08:01	

Examining the process names in the above table, several stand out. Among them are *TSVNCache.exe*, *imapi.exe*, *WPFFontCache_v0*, *cleansweep.exe* and *gmer.exe*. However, process *cleansweep.exe* stands out from the others (highlighted in red above). Internet searches for filenames *TSVNCache.exe* reveals it is the SVN cache process, while *imapi.exe* is an integral part of the Windows operating system and *WPFFontCache_v0* is the Windows WPF Font cache service. Finally, *gmer.exe* is a free rootkit and hidden application detection program while *cleansweep.exe* is related to a legacy Windows application used to clean Windows systems of accumulated debris.

Four of these five programs are considered "normal" whereas the existence of *cleansweep.exe* is altogether abnormal. A closer inspection is warranted concerning this process because it should not be found on this system.

3.3.1.3 Psscan plugin

The *psscan* plugin uses physical memory addresses and scans memory images for _EPROCESS pool allocations, in contrast to the *pslist* plugin that uses virtual memory addresses and scans for EPROCESS lists. The benefit of using this plugin is that sometimes it can succeed in listing processes that cannot be found using any of the other process listing plugins (i.e., *pslist* and *pstree*).

Consider the following output from this plugin, using command "*volatility -f spyeye.vmem psscan*":

Table 17: Volatility output for the Psscan plugin sorted by PID (SpyEye).

Offset(P)	Name	PID	PPID	PDB	Time created	Time exited
0x025c8830	System	4	0	0x00319000		
0x0236d7a0	wuauclt.exe	536	1068	0x0a9402c0	2010-11-11 22:03:33	
0x023fe020	smss.exe	572	4	0x0a940020	2010-11-11 22:02:08	
0x02503220	csrss.exe	636	572	0x0a940040	2010-11-11 22:02:13	
0x01f4c550	winlogon.exe	660	572	0x0a940060	2010-11-11 22:02:14	
0x0207d5f0	services.exe	704	660	0x0a940080	2010-11-11 22:02:15	
0x024264c0	lsass.exe	716	660	0x0a9400a0	2010-11-11 22:02:15	
0x0230c5f8	vmacthlp.exe	872	704	0x0a9400c0	2010-11-11 22:02:16	
0x0226cda0	svchost.exe	904	704	0x0a9400e0	2010-11-11 22:02:16	
0x023f2020	svchost.exe	972	704	0x0a940100	2010-11-11 22:02:17	
0x023e32f8	explorer.exe	1008	680	0x0a940320	2010-11-11 22:02:55	
0x024578b0	imapi.exe	1040	704	0x0a940220	2010-11-11 22:03:54	
0x022a0758	svchost.exe	1068	704	0x0a940120	2010-11-11 22:02:17	
0x01f4b020	svchost.exe	1108	704	0x0a940140	2010-11-11 22:02:17	
0x02406da0	svchost.exe	1232	704	0x0a940160	2010-11-11 22:02:18	
0x01ec2020	TSVNCache.exe	1252	1008	0x0a940340	2010-11-11 22:02:58	
0x02436a48	spoolsv.exe	1456	704	0x0a9401a0	2010-11-11 22:02:19	
0x01ebd300	VMwareTray.exe	1484	1008	0x0a940180	2010-11-11 22:03:00	
0x02067858	svchost.exe	1540	704	0x0a9401c0	2010-11-11 22:02:26	
0x02159958	VMwareUser.exe	1588	1008	0x0a9402e0	2010-11-11 22:03:00	
0x02072660	jqs.exe	1612	704	0x0a940200	2010-11-11 22:02:27	
0x0214ba18	jusched.exe	1672	1008	0x0a940300	2010-11-11 22:03:00	
0x02284b80	vmtoolsd.exe	1816	704	0x0a940240	2010-11-11 22:02:30	
0x022e69f8	VMUpgradeHelper	1872	704	0x0a940260	2010-11-11 22:02:30	
0x02458020	alg.exe	2108	704	0x0a940360	2010-11-11 22:03:54	
0x02226b48	cleansweep.exe	2268	1008	0x0a940460	2011-01-06 14:36:52	2011-01-06 14:36:52

Offset(P)	Name	PID	PPID	PDB	Time created	Time exited
0x020bd760	gmer.exe	2728	1008	0x0a9403a0	2011-01-06 14:37:41	
0x02389020	wscntfy.exe	2772	1068	0x0a940380	2010-11-11 22:03:56	
0x01ed9b50	wmiprvse.exe	2912	888	0x0a3004c0	2010-11-11 21:57:43	
0x01f7a708	WPFFontCache_v0	3084	704	0x0a940400	2010-11-11 22:05:04	
0x01f5e020	jucheck.exe	3892	1672	0x0a9402a0	2010-11-11 22:08:01	

The output from the *psscan* plugin appears very similar to the output of the *pslist* plugin. The suspicious process *cleansweep.exe* has been highlighted in red. At first glance differentiating between this output and that of *pslist* is not be apparent.

3.3.1.4 Differentiating the output between the pslist and psscan plugins

Highlighting the differences between the output from the *pslist* and *psscan* plugins may not be obvious at first glance. For this task, shell-based text processing is of significant use. By using the following command, it is readily possible to differentiate between the two plugins' output:

```
$ cat pslist.txt psscan.txt | awk '{print $2"\t"$3}' | sort |
uniq -c | grep -v "    2"
```

This command results in the following output:

```
1 --------------------  ------
1 ----------------      ------
1 wmiprsve.exe    2912
```

Thus, by using these commands, it was determined that the difference between these two plugins (*pslist* and *psscan*) is process *wmiprvse.exe*, a normal Windows process that is often not visible using standard process listings (i.e. *pslist*).

3.3.1.5 Psxview plugin

Volatility provides an additional capability for detecting hidden running processes. The *psxview* plugin provides a detailed listing of processes running in a memory image by using five specific detection methods. These include *pslist*, *psscan*, *thrdproc*, *pspcdid* and *csrss*.

Consider the following output from this plugin, using command "*volatility -f spyeye.vmem psxview*":

Table 18: Volatility output for the Psxview plugin sorted by PID (SpyEye).

Offset(P)	Name	PID	pslist	psscan	thrdproc	pspcdid	csrss
0x025c8830	System	4	True	True	True	True	False
0x0236d7a0	wuauclt.exe	536	True	True	True	True	True
0x023fe020	smss.exe	572	True	True	True	True	False
0x02503220	csrss.exe	636	True	True	True	True	False

Offset(P)	Name	PID	pslist	psscan	thrdproc	pspcdid	csrss
0x01f4c550	winlogon.exe	660	True	True	True	True	True
0x0207d5f0	services.exe	704	True	True	True	True	True
0x024264c0	lsass.exe	716	True	True	True	True	True
0x0230c5f8	vmacthlp.exe	872	True	True	True	True	True
0x0226cda0	svchost.exe	904	True	True	True	True	True
0x023f2020	svchost.exe	972	True	True	True	True	True
0x023e32f8	explorer.exe	1008	True	True	True	True	True
0x024578b0	imapi.exe	1040	True	True	True	True	True
0x022a0758	svchost.exe	1068	True	True	True	True	True
0x01f4b020	svchost.exe	1108	True	True	True	True	True
0x02406da0	svchost.exe	1232	True	True	True	True	True
0x01ec2020	TSVNCache.exe	1252	True	True	True	True	True
0x02436a48	spoolsv.exe	1456	True	True	True	True	True
0x01ebd300	VMwareTray.exe	1484	True	True	True	True	True
0x02067858	svchost.exe	1540	True	True	True	True	True
0x02159958	VMwareUser.exe	1588	True	True	True	True	True
0x02072660	jqs.exe	1612	True	True	True	True	True
0x0214ba18	jusched.exe	1672	True	True	True	True	True
0x02284b80	vmtoolsd.exe	1816	True	True	True	True	True
0x022e69f8	VMUpgradeHelper	1872	True	True	True	True	True
0x02458020	alg.exe	2108	True	True	True	True	True
0x02226b48	cleansweep.exe	2268	True	True	False	True	False
0x020bd760	gmer.exe	2728	True	True	True	True	True
0x02389020	wscntfy.exe	2772	True	True	True	True	True
0x01ed9b50	wmiprvse.exe	2912	False	True	False	False	False
0x01f7a708	WPFFontCache_v0	3084	True	True	True	True	True
0x01f5e020	jucheck.exe	3892	True	True	True	True	True

Note that some processes listed as hidden using the *csrss* method are not always hidden. For Windows 7 and Vista systems, the list of internal processes is not available and in certain cases where Windows XP is concerned, required memory pages may have been swapped out, thereby affecting the outcome. [19]

However, what is not normal is that process *wmiprvse.exe* (PID 2912) is hidden from all *psxview*-based memory detection methods except for *psscan*. This may be indicative of a process having

been previously terminated but the system process table does not correctly reflect the fact. This must, however, be confirmed. Finally, based on its name, it is likely a legitimate Windows system process.

Interestingly, process *cleansweep.exe* (PID 2268) is only hidden from *thrdproc* and *csrss*. Since this process makes no effort to hide itself from observation it is likely that this process in of itself is not itself infected meaning that injected code may be lurking somewhere in this memory image. It is expected that an advanced Trojan like SpyEye would make efforts to conceal itself.

3.3.1.6 Correlating PIDs and PPIDs***

Examining the output established thus far based on the *pslist*, *psscan* and *psxview* plugins, the following information can be established with respect to process instantiation.

Table 19: Process instantiation for suspicious processes (SpyEye)

PPID name	PPID	PID	PID name
N/A	888	2912	wmiprvse.exe
N/A	680	1008	explorer.exe
explorer.exe	1008	2268	cleansweep.exe

Thus, it can be inferred that from process *explorer.exe* (PID 1008), process *cleansweep.exe* (PID 2268) was instantiated. It is not known what instantiated PPID 888.

The next step is to determine if other plugins can reveal evidence of infection.

3.3.2 Step 2: Assess other sources of evidence

This step examines various Volatility plugins that can be used to establish additional evidence concerning the memory image.

3.3.2.1 Cmdscan and consoles plugins

The plugins *cmdscan* and *consoles* plugin may reveal more information about commands typed into a command shell. Querying a memory image using these two plugins is carried out using the following commands:

```
$ volatility -f spyeye.vmem cmdscan

$ volatility -f spyeye.vmem consoles
```

These *cmdscan* plugin revealed absolutely no information whatsoever while the *consoles* plugin provided output concerning two console-based processes that are unrelated to this investigation.

3.3.2.2 Connscan plugin

The first network-based Volatility plugin that should be used is *connscan*. It is used to verify the existence of ongoing network connections and it scans a memory image for current or recently terminated connections.

Consider the following output from this plugin, using command "*volatility -f spyeye.vmem connscan*":

Table 20: Volatility output for the Connscan plugin (SpyEye)

Offset(P)	Local Address	Remote Address	PID
0x01eacc00	192.168.16.129:1039	65.55.185.26:443	1068
0x01fd3170	192.168.16.129:1040	207.46.21.58:80	1068

Based on this information, PID 1068 (*svchost.exe*) is communicating with two remote systems, 65.55.185.26 and 207.46.21.58. A Whois search for these two systems confirms that they both belong to Microsoft. However, DNS name resolution is not possible for these systems at this time.

Thus, whatever network traffic was exchanged between these systems does not appear to be related to the infection. It could be a Windows update, although this is only conjecture.

3.3.2.3 Connections plugin

The *connections* plugin can be used to determine information concerning recently terminated and ongoing communications. It therefore makes sense to use this plugin to query a memory image for additional network-based information.

However, using command "*volatility -f spyeye.vmem connections*" yielded no output whatsoever.

3.3.2.4 Sockets and sockscan plugins

Volatility offers two additional network-based plugins, *sockets* and *sockscan*. The *sockets* plugin lists open sockets that may provide additional information about covert network channels, while the *sockscan* plugin scans a suspect memory image for all TCP sockets. Generally, the output is the same for both plugins with the exception of memory addresses, where the *sockets* plugin uses virtual memory addresses while the *sockscan* plugin uses physical memory addressing.

Thus, using the following commands it will be possible to determine which processes have open network sockets ready for communications:

```
$ volatility -f spyeye.vmem sockets > sockets.txt
$ volatility -f spyeye.vmem sockscan > sockscan.txt
```

```
$ cat sockets.txt sockscan.txt | sort | awk '{$1="";print}'
| uniq > sockets_sockscan.txt
```

The output of file *sockets_sockscan.txt* appears as shown in the following table:

Table 21: Volatility Sockets and Sockscan plugin output sorted by PID (SpyEye)

PID	Port	Proto	Protocol	Address	Create Time
4	445	17	UDP	0.0.0.0	2010-11-11 22:02:08
4	445	6	TCP	0.0.0.0	2010-11-11 22:02:08
4	445	17	UDP	0.0.0.0	2010-11-11 22:02:08
4	445	6	TCP	0.0.0.0	2010-11-11 22:02:08
716	500	17	UDP	0.0.0.0	2010-11-11 22:02:27
716	4500	17	UDP	0.0.0.0	2010-11-11 22:02:27
716	0	255	Reserved	0.0.0.0	2010-11-11 22:02:27
716	500	17	UDP	0.0.0.0	2010-11-11 22:02:27
716	4500	17	UDP	0.0.0.0	2010-11-11 22:02:27
716	0	255	Reserved	0.0.0.0	2010-11-11 22:02:27
972	135	6	TCP	0.0.0.0	2010-11-1122:02:17
972	135	6	TCP	0.0.0.0	2010-11-11 22:02:17
1068	123	17	UDP	127.0.0.1	2011-01-06 14:36:59
1068	123	17	UDP	127.0.0.1	2011-01-06 14:36:59
1232	1900	17	UDP	127.0.0.1	2011-01-06 14:36:59
1232	1900	17	UDP	127.0.0.1	2011-01-06 14:36:59
1612	5152	6	TCP	127.0.0.1	2010-11-11 22:02:27
1612	5152	6	TCP	127.0.0.1	2010-11-11 22:02:27
2108	1025	6	TCP	127.0.0.1	2010-11-11 22:03:54
2108	1025	6	TCP	127.0.0.1	2010-11-11 22:03:54
3892	1026	6	TCP	0.0.0.0	2010-11-11 22:08:01
3892	1026	6	TCP	0.0.0.0	2010-11-11 22:08:01

Looking at this data, based on the list of open sockets, none of the listed communications corresponds to the already established PIDs of interest (2268 and 2912). Examining the list of ports, it is suspicious that PID 1068 (*explorer.exe*) has an open socket for port 123 (NTP). The valid use of this port is not typically characteristic of *explorer.exe*. Thus, this behaviour is highly suspicious.

3.3.2.5 Filescan plugin

If an infection is active and does not show itself via the network then the *filescan* plugin may be of assistance as the plugin may be able to find open file handles in memory. Unfortunately, no direct link to these handles is possible as the physical disk image is not available for analysis. This plugin makes use of physical address offsets.

The preferred method for detecting indicators of compromise is twofold. First, using keywords (e.g. SpyEye, infection, rootkit, worm, etc.) it may be possible to find the infection, as malware programmers do not often use innocuous looking filenames. Of course, this is at best a hit and miss approach. Secondly, it can be attempted to detect suspicious files based on their locations. However, this requires that the investigator has a very good working knowledge of the underlying operating system as just looking at filenames[3] and locations will not produce meaningful results, unless something really sticks out.

Fortunately, as with the Zeus infection ([22]), much useful information abounds for the SpyEye Trojan as listed in Section 3.1. Thus, sufficient potential keywords can be found.

Running command "*volatility -f spyeye.vmem filescan | grep -i -P '(botnet|cleansweep|spyeye|virus|Trojan|rootkit|worm|jusched|config|php|recycle)'* " results in the following output pertinent output, after manual pruning:

```
0x0203ceb8                    1                    0   R--r-d
\Device\HarddiskVolume1\cleansweep.exe\cleansweep.exe

0x02072f10                    1                    0   RW-rw-
\Device\HarddiskVolume1\Documents                      and
Settings\Administrator\Desktop\spyeye\spyeye\cf.bin

0x020747a8                    1                    0   R--r--
\Device\HarddiskVolume1\Documents                      and
Settings\Administrator\Desktop\spyeye_sample.piz

0x021292e8                    1                    0   R--rwd
\Device\HarddiskVolume1\Documents                      and
Settings\Administrator\Desktop\spyeye\spyeye\intro-
spyeye.pdf

0x02265ad8                    1                    0   R--rwd
\Device\HarddiskVolume1\Documents                      and
Settings\Administrator\Desktop\spyeye\spyeye\cs.idb

0x0229fd38                    1                    0   R--rwd
\Device\HarddiskVolume1\Documents                      and
Settings\Administrator\Desktop\spyeye\spyeye\edc7c152759ba0
482bd39db0ea2c4319
```

[3] Recall that a reliable source of filenames is the NSRL hash-set. It can be broken down by software product and operating system.

```
0x02306b18                    1                    0    RW-rw-
\Device\HarddiskVolume1\Documents                       and
Settings\Administrator\Desktop\spyeye\spyeye\config.bin.zip

0x02307148                    1                    0    -W-rwd
\Device\HarddiskVolume1\cleansweep.exe\config.bin

0x02362650                    1                    0    R--rwd
\Device\HarddiskVolume1\Documents                       and
Settings\Administrator\Desktop\spyeye\spyeye\cs.exe

0x02401f90                    1                    0    R--rwd
\Device\HarddiskVolume1\Documents                       and
Settings\Administrator\Desktop\spyeye\spyeye\SymantecViralP
ortalDownloadReport.txt

0x02524250                    1                    0    R--rwd
\Device\HarddiskVolume1\Documents                       and
Settings\Administrator\Desktop\spyeye\spyeye\2b8a408b56eaf3
ce0198c9d1d8a75ec0
```

Based on this output and established documentation [6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17 and 18], a convincing argument can be made that PID 2268 (*cleansweep.exe*) is linked to the SpyEye Trojan horse. Furthermore, it appears that the various SpyEye configuration files have been found.

Finally, no file handles were found for process PID 2912 (*wmiprvse.exe*) indicating that it is likely this process previously terminated and that the system process table was not correctly updated.

3.3.2.6 Mutantscan plugin

The Volatility *mutantscan* can sometimes reveal interesting information about Windows thread-based mutexes in memory. It makes use of physical offset addressing.

Using command "*volatility -f spyeye.vmem mutantscan*" yielded the following pertinent information after pruning the output (as the output is several pages long):

```
0x02039b78          2          1                1   0x00000000
__CLEANSWEEP__
```

The above output indicates that a suspicious mutex for was found. The mutex was most likely put in place by suspicious process *cleansweep.exe*.

3.3.2.7 Handles plugin

The Volatility *handles* plugin can reveal interesting information about processes and resources attached or associated to them that might not be found using previously examined plugins. It makes use of virtual offset addressing.

Using command "*volatility -f spyeye.vmem handles*," the following pruned output is of interest to the investigation and is as follows:

```
0x82039b78      1008            0x5b4      0x1f0001  Mutant
__CLEANSWEEP__

0x82226b48      1008            0x6d0      0x1f0fff  Process
cleansweep.exe(2268)

0x82522470      1008            0x608      0x1f03ff  Thread
TID 2216 PID 2268
```

From this output, it can be determined that there is a link between PID 1008 (*explorer.exe*) and 2268 (*cleansweep.exe*). Moreover, a mutex handle was discovered for PID 1008 with value of *__CLEANSWEEP__* that is highly suspicious, as this should never be found in process *explorer.exe*. Finally, PID 2268 is associated with TID 2216.

Note that the mutex has virtual memory address *0x82039b78* while the mutex uncovered by the mutantscan plugin, listed using a physical memory address, has an address of *0x02039b78*. These are in fact the very same mutex.

3.3.2.8 Threads plugin

The final Volatility plugin to be used in this step is the *threads* plugin. Armed with the information provided by the *handles* plugin, it is worthwhile investigating the information uncovered about TID 2216. Using command "*volatility -f spyeye.vmem threads -p 2268*" yielded the following information:

```
ETHREAD: 0x82522470 Pid: 2268 Tid: 2216
Tags:
Created: 2011-01-06 14:36:52
Exited: 2011-01-06 14:36:52
Owning Process: cleansweep.exe
Attached Process: cleansweep.exe
State: Terminated
BasePriority: 0x8
Priority: 0x10
TEB: 0x00000000
StartAddress: 0x7c810705 UNKNOWN
ServiceTable: 0x80552fe0
  [0] 0x80501bbc
  [1] 0xbf99b400
  [2] 0x00000000
  [3] 0x00000000
Win32Thread: 0x00000000
CrossThreadFlags: PS_CROSS_THREAD_FLAGS_TERMINATED
```

Thus, suspicious thread TID 2216 is without doubt a subset of process PID 2268.

3.3.3 Step 3: Dump and assess suspicious processes

The evidence established thus far indicates that two processes are suspicious, PIDs 2268 and 2912 (*cleansweep.exe* and *wmiprvse.exe*). However, PID 2912 is likely the remains of a terminated process where the system process table was incorrectly updated. Nevertheless, both processes will be evaluated in this step for potential infection.

3.3.3.1 Create data directories

Create directories *malfind*, *memdump*, *procexedump* and *procmemdump* for storing memory samples dumped from the memory image using corresponding Volatility plugins. This is done using the following commands:

```
$ mkdir malfind

$ mkdir memdump

$ mkdir procexedump

$ mkdir procmemdump
```

3.3.3.2 Malfind plugin

3.3.3.2.1 Running the plugin

Volatility's *malfind* plugin was specifically designed to search for malware hidden through code injection. If memory address offsets are specified then they must be physical memory address offsets.

Using the following commands it was attempted to find and dump injected code associated with *cleansweep.exe* (PID 2268) and *wmiprvse.exe* (PID 2912):

```
$ volatility -f spyeye.vmem malfind -p 2268 -o 0x02226b48
--dump-dir=malfind

$ volatility -f spyeye.vmem malfind -p 2912 -o 0x01ed9b50
--dump-dir=malfind
```

These two commands resulted in no output thereby confirming that processes *cleansweep.exe* and *wmiprvse.exe* were themselves not injected with malicious code.

Using the plugin at large against the infected memory image with command "*volatility -f spyeye.vmem malfind*" resulted in the dumping of 36 memory samples. A great deal of output was generated by the plugin.

Detailed analyses of the findings are carried out in subsequent sections.

3.3.3.2.2 Scanning the dumped memory samples

All 36 samples were then scanned using the six aforementioned scanners resulting in the following dumped memory samples having been determined as potentially malicious or infected:

Table 22: Scanner infection detection for dumped memory samples from the Malfind plugin (SpyEye)

Scanner	Filename	Infection Identification
Avast	process.0x823e32f8.0xea00000.dmp	Win32:Malware-gen
AVG	process.0x823e32f8.0xea00000.dmp	Trojan horse Generic28.BLVW
BitDefender	N/A	Nothing found
ClamAV	N/A	Nothing found
FRISK	N/A	Nothing found
McAfee	process.0x81ebd300.0xea50000.dmp	Generic.dx!D2309E0CF132 trojan
	process.0x81ec2020.0xea50000.dmp	Generic.dx!2C068D497643 trojan
	process.0x81f4b020.0xea50000.dmp	Generic.dx!D2309E0CF132 trojan
	process.0x81f4c550.0xea50000.dmp	Generic.dx!DF17D637CA65 trojan
	process.0x81f5e020.0xea50000.dmp	Generic.dx!2C068D497643 trojan
	process.0x82067858.0xea50000.dmp	Generic.dx!2C068D497643 trojan
	process.0x82072660.0xea50000.dmp	RDN/Generic.dx!ccs trojan
	process.0x8207d5f0.0xea50000.dmp	Generic.dx!D2309E0CF132 trojan
	process.0x8214ba18.0xea50000.dmp	Generic.dx!2C068D497643 trojan
	process.0x82159958.0xea50000.dmp	Generic.dx!2C068D497643 trojan
	process.0x8226cda0.0xea50000.dmp	Generic.dx!D2309E0CF132 trojan
	process.0x82284b80.0xea50000.dmp	Generic.dx!2C068D497643 trojan
	process.0x822a0758.0xea50000.dmp	Generic.dx!2C068D497643 trojan
	process.0x822e69f8.0xea50000.dmp	Generic.dx!2C068D497643 trojan
	process.0x823e32f8.0xea50000.dmp	Generic.dx!2E94730C0F5D trojan
	process.0x823e32f8.0xeab0000.dmp	Generic.dx!B6ED55D2C36B trojan
	process.0x823f2020.0xea50000.dmp	Generic.dx!D2309E0CF132 trojan
	process.0x82406da0.0xea50000.dmp	Generic.dx!D2309E0CF132 trojan
	process.0x824264c0.0xea50000.dmp	Generic.dx!D2309E0CF132 trojan
	process.0x82436a48.0xea50000.dmp	Generic.dx!D2309E0CF132 trojan
	process.0x82458020.0xea50000.dmp	Generic.dx!D2309E0CF132 trojan

Based on these results, the reader must consider that the results offered by McAfee are likely false positives as none of the other five scanners picked up on these dumped files. However, both

AVG and Avast detected dumped file *process.0x823e32f8.0xea00000.dmp* as infected. Looking more closely at this file using the Linux *file* command revealed that the file was a UPX-based executable. However, examining it in a hex editor quickly exposed it is as the Trojan horse of interest based on specific strings found therein:

- `*Dropper*!main : CreateMutex->ERROR_ALREADY_EXISTS`
- `*Dropper* : BOT_VERSION = %d, PID = %d, szModuleFileName = "%s"`

Moreover, the executable was not actually UPX-based but instead merely had UPX headers occupying the first several kilobytes of the dumped memory sample. This same analytical process was applied to all the executables detected by McAfee as infected but they were all found to be innocuous.

3.3.3.2.3 SHA1 and fuzzy hashes

All 36 dumped files were hashed using the *sha1sum* command to generate their SHA1 signatures. Based on these hashes, it was determined that only 15 unique signatures existed indicating that the remaining memory samples were duplicates. Specifically, the unique hash signatures, sorted alphanumerically are:

0a32ad8919f968283fe100fa1c13f830095c673d
20809106fd5d6dcf512967cb2ee3444e26a77719
3870c69b74fca8eed2cdb5e91b6a02b9b96850ac
5f56930b5a5d1e813121a8f037d2cdfa7b639433
6899aefcec4330f28e6908f85dcbe1db39568ef7
7c0af938ac30ac6ba3e5860165c221c3ac899cab
86a00f1b6f6028174c67d8752bf9056f51c1e7f8
8a6e4adc3ecc502dea4b567b81f8c0194af17a37
a516cd767985107043e88217a536b745b00ef67a
a73d4ec7ef7c287cb4e358857dc0600a6cbd6d0d
a9be8e7823581cbf75f921a8076051ce135a2c9a
b580eac5ae22126dd4cfdb90246903a2394e31f2
d276dfe740acf3582332bb34456b98dd76074872
e02f2f4d613cac595db604940b5cea03dbb64d4e
efe49e97081056804f46770960505f46cc90f356

A complete SHA1-filename listing is available in the following table:

Table 23: SHA1 vs. filename for Malfind dumped memory samples (SpyEye)

SHA1 Hash	Filename
0a32ad8919f968283fe100fa1c13f830095c673d	process.0x81f7a708.0xea50000.dmp[4]
0a32ad8919f968283fe100fa1c13f830095c673d	process.0x8230c5f8.0xea50000.dmp
0a32ad8919f968283fe100fa1c13f830095c673d	process.0x82389020.0xea50000.dmp
0a32ad8919f968283fe100fa1c13f830095c673d	process.0x824578b0.0xea50000.dmp
20809106fd5d6dcf512967cb2ee3444e26a77719	process.0x823e32f8.0x26b0000.dmp
3870c69b74fca8eed2cdb5e91b6a02b9b96850ac	process.0x820bd760.0xeab0000.dmp
5f56930b5a5d1e813121a8f037d2cdfa7b639433	process.0x823e32f8.0xea00000.dmp
6899aefcec4330f28e6908f85dcbe1db39568ef7	process.0x8236d7a0.0xea50000.dmp
7c0af938ac30ac6ba3e5860165c221c3ac899cab	process.0x81ec2020.0xea50000.dmp
7c0af938ac30ac6ba3e5860165c221c3ac899cab	process.0x81f5e020.0xea50000.dmp
7c0af938ac30ac6ba3e5860165c221c3ac899cab	process.0x82067858.0xea50000.dmp
7c0af938ac30ac6ba3e5860165c221c3ac899cab	process.0x8214ba18.0xea50000.dmp
7c0af938ac30ac6ba3e5860165c221c3ac899cab	process.0x82159958.0xea50000.dmp
7c0af938ac30ac6ba3e5860165c221c3ac899cab	process.0x82284b80.0xea50000.dmp
7c0af938ac30ac6ba3e5860165c221c3ac899cab	process.0x822a0758.0xea50000.dmp
7c0af938ac30ac6ba3e5860165c221c3ac899cab	process.0x822e69f8.0xea50000.dmp
86a00f1b6f6028174c67d8752bf9056f51c1e7f8	process.0x820bd760.0x13d0000.dmp
86a00f1b6f6028174c67d8752bf9056f51c1e7f8	process.0x820bd760.0x15d0000.dmp
86a00f1b6f6028174c67d8752bf9056f51c1e7f8	process.0x820bd760.0x19d0000.dmp
86a00f1b6f6028174c67d8752bf9056f51c1e7f8	process.0x820bd760.0x1bd0000.dmp
8a6e4adc3ecc502dea4b567b81f8c0194af17a37	process.0x82503220.0x7f6f0000.dmp
a516cd767985107043e88217a536b745b00ef67a	process.0x82072660.0xea50000.dmp
a73d4ec7ef7c287cb4e358857dc0600a6cbd6d0d	process.0x823e32f8.0xea50000.dmp
a9be8e7823581cbf75f921a8076051ce135a2c9a	process.0x81f4c550.0xea50000.dmp
b580eac5ae22126dd4cfdb90246903a2394e31f2	process.0x820bd760.0xd20000.dmp
d276dfe740acf3582332bb34456b98dd76074872	process.0x81ebd300.0xea50000.dmp
d276dfe740acf3582332bb34456b98dd76074872	process.0x81f4b020.0xea50000.dmp

[4] Consider that filename *process.0x81f7a708.0xea50000.dmp* denotes that the process dumped was located within a process with a virtual memory address of *0x81f7a708* and was found at an address of *0xea50000* therein.

SHA1 Hash	Filename
d276dfe740acf3582332bb34456b98dd76074872	process.0x8207d5f0.0xea50000.dmp
d276dfe740acf3582332bb34456b98dd76074872	process.0x8226cda0.0xea50000.dmp
d276dfe740acf3582332bb34456b98dd76074872	process.0x823f2020.0xea50000.dmp
d276dfe740acf3582332bb34456b98dd76074872	process.0x82406da0.0xea50000.dmp
d276dfe740acf3582332bb34456b98dd76074872	process.0x824264c0.0xea50000.dmp
d276dfe740acf3582332bb34456b98dd76074872	process.0x82436a48.0xea50000.dmp
d276dfe740acf3582332bb34456b98dd76074872	process.0x82458020.0xea50000.dmp
e02f2f4d613cac595db604940b5cea03dbb64d4e	process.0x81f5e020.0xeab0000.dmp
efe49e97081056804f46770960505f46cc90f356	process.0x823e32f8.0xeab0000.dmp

Dumped memory sample file *0x823e32f8.0xea00000.dmp* confirmed with malicious code has been highlighted in red in the above table.

A full listing of fuzzy hashes obtained against the 36 dumped memory samples can be found in Annex E.

Comparing the 15 unique SHA1 hashes against the NSRL hash-set found no matches. Furthermore, comparing these hashes against the SHA1 hashes of the carved data files also found no matches.

Comparing the fuzzy hashes of the *malfind*-dumped memory samples against dumped file *process.0x823e32f8.0xea00000.dmp* (designated as infected by both AVG and Avast) indicated no matches, not even partially. When comparing the fuzzy hashes of the *malfind*-dumped memory samples against the fuzzy hashes of the carved data files, one partial match was obtained, specifically a 41% match between *malfind*-dumped file *process.0x820bd760.0xd20000.dmp* and carved data file *f0306064.dll*.

Carved data file *f0306064.dll* was heuristically detected as malicious by F-Prot. The specific scanner log can be found in Annex A.2.5. However, a vigilant *strings* analysis revealed that f0306064.fll and *process.0x820bd760.0xd20000.dmp* were innocuous and of no further concern.

3.3.3.2.4 Explorer.exe malfind output

Based on the previous scanner results, it can be established that file *process.0x823e32f8.0xea00000.dmp* (see Table 16) is indicative of *explorer.exe* (PID 1008) as proposed in Section 3.3.3.2.2. Consider that this process had virtual memory address *0x823e32f8* and that sub address *0xea00000* corresponds to the memory address of the malicious injected code within *explorer.exe*'s memory space.

The *malfind* output specific to memory sample *process.0x823e32f8.0xea00000.dmp* was as follows:

```
Process: explorer.exe Pid: 1008 Address: 0xea00000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 43, MemCommit: 1, PrivateMemory: 1,
Protection: 6

0x0ea00000  4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00
MZ..............
0x0ea00010  b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00
........@.......
0x0ea00020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
................
0x0ea00030  00 00 00 00 00 00 00 00 00 00 00 00 f8 00 00 00
................

0xea00000 4d              DEC EBP
0xea00001 5a              POP EDX
0xea00002 90              NOP
0xea00003 0003            ADD [EBX], AL
0xea00005 0000            ADD [EAX], AL
0xea00007 000400          ADD [EAX+EAX], AL
0xea0000a 0000            ADD [EAX], AL
0xea0000c ff              DB 0xff
0xea0000d ff00            INC DWORD [EAX]
0xea0000f 00b800000000    ADD [EAX+0x0], BH
0xea00015 0000            ADD [EAX], AL
0xea00017 004000          ADD [EAX+0x0], AL
0xea0001a 0000            ADD [EAX], AL
0xea0001c 0000            ADD [EAX], AL
0xea0001e 0000            ADD [EAX], AL
0xea00020 0000            ADD [EAX], AL
0xea00022 0000            ADD [EAX], AL
0xea00024 0000            ADD [EAX], AL
0xea00026 0000            ADD [EAX], AL
0xea00028 0000            ADD [EAX], AL
0xea0002a 0000            ADD [EAX], AL
0xea0002c 0000            ADD [EAX], AL
0xea0002e 0000            ADD [EAX], AL
0xea00030 0000            ADD [EAX], AL
0xea00032 0000            ADD [EAX], AL
0xea00034 0000            ADD [EAX], AL
0xea00036 0000            ADD [EAX], AL
0xea00038 0000            ADD [EAX], AL
0xea0003a 0000            ADD [EAX], AL
0xea0003c f8              CLC
0xea0003d 0000            ADD [EAX], AL
0xea0003f 00              DB 0x0
```

3.3.3.2.5 Summary

Running the *malfind* plugin, it was not expected that it would have determined that 36 processes (and threads) had potentially injected code in them. Thus, the only way to determine which were of interest was to scan them in the hopes of detecting possibly malicious or infected code and also establish similarities between files based on their SHA1 and fuzzy hashes.

Thus far, it has been established that process *explorer.exe* was injected with malicious code. As for *cleansweep.exe* and *wmiprvse.exe*, though nothing has been found thus far, they cannot yet be excluded from further analysis.

3.3.3.3 Memdump plugin

3.3.3.3.1 Running the plugin

The *memdump* plugin is used to dump a process' addressable memory space. If memory address offsets are specified then they must be physical memory addresses. The plugin dumps all data segments associated with a specified process to a destination file. Moreover, the plugin is worth trying as the information attained from it, combined with any potential information obtained from subsequent plugins, can be used in the reverse engineering of the malware.

The commands used to dump the addressable memory space of processes *explorer.exe* (PID 1008), *cleansweep.exe* (PID 2268) and *wmiprvse.exe* (PID 2912), were:

```
$ volatility -f ../../spyeye.vmem memdump -p 1008 -o
0x023e32f8 --dump-dir=memdump

$ volatility -f ../../spyeye.vmem memdump -p 2268 -o
0x02226b48 --dump-dir=memdump

$ volatility -f ../../spyeye.vmem memdump -p 2912 -o
0x01ed9b50 --dump-dir=memdump
```

This resulted in two dumped memory samples, one for PID 1008 (*explorer.exe*) and another for PID 2268 (*cleansweep.exe*). Nothing was dumped for PID 2912 (*wmiprvse.exe*). The two memory samples had the following metadata:

Table 24: Metadata for PID 1008 dumped using the Memdump plugin (SpyEye)

Filename	memdump/1008.dmp
Size	174,731,264 bytes
SHA1 hash	a9998d03c21fc7c216cd20784a5e95eb23b826c3
Fuzzy hash	3145728:9dVyCvqB0d3sRC/FKtK1Je9W54egVPWOBYC077f29IMzJCbmOB8AW12gc4+JOtjj:9dVyLC/FKtK1Je9W54egVPWOBYC077fC

Table 25: Metadata for PID 2268 dumped using the Memdump plugin (SpyEye)

Filename	memdump/2268.dmp
Size	137,220,096 bytes
SHA1 hash	fdcb9a51fc62168038222fdd758433888ec152a5
Fuzzy hash	3145728:2d3sRC/FKtK1Je9W54egVPWOBYC077f29IMzJCbmOB8AW1Xgc4+JOtjBi2XmuPVI:ZC/FKtK1Je9W54egVPWOBYC077f29IMj

All file metadata was saved for potential future use.

3.3.3.3.2 Virus scanning and file hashing

As only two memory samples were dumped using the plugin, scanning and hashing was straightforward. All six scanners found that the dumped two files, *memdump/1008.dmp* and *memdump/2268.dmp*, were uninfected.

However, a *strings*-based 7, 8, 16 and 32-bit examination of file *memdump/1008.dmp* found that it contained malicious strings, the same as found with the *malfind* plugin:

```
*Dropper*!main : CreateMutex->ERROR_ALREADY_EXISTS

*Dropper* : BOT_VERSION = %d, PID = %d, szModuleFileName =
"%s"
```

Thus, it can be readily concluded that the dumped memory sample for process PID 1008 contained not only all the code and data relating to *explorer.exe* but also the maliciously injected code.

A thorough 7, 8, 16 and 32-bit *strings* examination of PID 2268 (*cleansweep.exe*) found no direct evidence of malicious code or strings.

Despite these facts, fuzzy hashing has revealed that the *memdump*-dumped files for PID 1008 and 2268 share an 83% match between one another, indicating that much of the data and code between them is similar.

Comparing their fuzzy hashes the *malfind*-dumped file did not reveal any matches, not even partially. Thus, it can be concluded that any correlation between them is too small to be of significance.

SHA1 hash matching between the two *memdump*-dumped files and the NSRL hast-set revealed no matches. Fuzzy hash matching between the two files and the carved data files also revealed no matches.

3.3.3.3.3 Summary

Although the *malfind* plugin specifically dumped the injected code associated with the SpyEye Trojan horse, the *memdump* plugin succeeded in dumping all memory and data segments associated with processes PID 1008 and 2268 (*explorer.exe* and *cleansweep.exe*, respectively).

Based on these two *memdump*-dumped files, it was possible to determine based on 7, 8, 16 and 32-bit *strings* analysis that no malicious code was apparent in file *2268.dmp*, thereby indicating that even though *cleansweep.exe* is associated with the Trojan horse, possibly as a dropper, it is not the actual Trojan horse.

However, malicious code was found relating to this Trojan horse within the memory space of PID 1008 (file *memdump/1008.dmp*), indicating that it contained not only the injected code detected by the *malfind* plugin relating to *explorer.exe* but that it also contained all the other data segments and executable code concerning *explorer.exe*.

3.3.3.4 Procexedump plugin

3.3.3.4.1 Running the plugin

Unlike the *memdump* plugin, the *procexedump* plugin dumps only a process' executable code. If memory addresses are specified then they must be physical memory offsets.

The commands used to dump the executable code for PIDs 1008, 2268 and 2912 (*explorer.exe*, *cleansweep.exe* and *wmiprvse.exe*, respectively) were:

```
$ volatility -f ../../spyeye.vmem procexedump -p 1008 -o 0x023e32f8 --dump-dir=procexedump

$ volatility -f ../../spyeye.vmem procexedump -p 2268 -o 0x02226b48 --dump-dir=procexedump

$ volatility -f ../../spyeye.vmem procexedump -p 2912 -o 0x01ed9b50 --dump-dir=procexedump
```

The second command generated the following error:

```
Process(V) ImageBase  Name                 Result
---------- ---------- -------------------- ------
0x82226b48 ---------- cleansweep.exe         Error: PEB at 0x7ffdd000 is paged
```

The third command generated the following error:

```
Process(V) ImageBase  Name                 Result
---------- ---------- -------------------- ------
---------- ---------- -------------------- Error: Cannot acquire process AS
```

These errors indicate that the memory space for *cleansweep.exe* has been paged out. From the second error, it can be concluded that PID 2912 is no longer available on this system and that the system process-based list table was not correctly updated.

Only one file was dumped as a result of these commands and it had the following metadata:

Table 26: Metadata for PID 1008 dumped using the Procexedump plugin (SpyEye)

Filename	procexedump/executable.1008.exe
Size	1,033,728 bytes
SHA1 hash	d8fee09f59ef07aa2b363d10e107e0f58e930d90
Fuzzy hash	12288:8HmcoCUyZtwAvAs4wTCyrPTaoHWYUrkf8w0Vnzac1/g/J/vMS:mmfty/wAvN7lrwbkf8w0VnH1/g/J/k

All file metadata was saved for possible future use.

3.3.3.4.2 Virus scanning and file hashing

The SHA1 hash of file *procexedump/executable.1008.exe* was compared against the SHA1 hashes of the NSRL hash-set but no matches were found.

Fuzzy hash matching was carried out comparing file *procexedump/executable.1008.exe* against the carved data files but no matches, even partial, could be established. Then, fuzzy hash matching was conducted comparing the *procexedump*-dumped memory sample against both the *malfind* and *memdump*-dumped memory samples but no matches were established.

The six scanners were used if the *procexedump*-dumped memory sample, was potentially infected but nothing was found.

A vigilant 7, 8, 16 and 32-bit *strings*-based examination of the memory sample failed to find anything suspicious with this file.

3.3.3.4.3 Summary

The *procexedump* plugin, while having succeeded in dumping the executable code for PID 1008 (*explorer.exe*), did not contain any of the injected code found using the *malfind* plugin. Moreover, the suspicion of process PID 2912 (*wmiprvse.exe*) has been laid to rest and should no longer be considered an issue.

3.3.3.5 Procmemdump plugin

3.3.3.5.1 Using the plugin

Unlike the *memdump* plugin, the *procmemdump* plugin dumps a process' executable code, including associated slack space (all processes have some slack space). The commands used to

do this for PIDs 1008, 2268 and 2912 (*explorer.exe*, *cleansweep.exe* and *wmiprvse.exe*, respectively) were:

```
$ volatility -f ../../spyeye.vmem procmemdump -p 1008 -o 0x023e32f8 --dump-dir=procexedump

$ volatility -f ../../spyeye.vmem procmemdump -p 2268 -o 0x02226b48 --dump-dir=procexedump
```

The second command generated the following error:

```
Process(V) ImageBase  Name                 Result
---------- ---------- -------------------- ------
0x82226b48 ---------- cleansweep.exe         Error: PEB at 0x7ffdd000 is paged
```

This error indicates that the memory space for *cleansweep.exe* has been paged out.

Only one file was dumped as a result of these commands and it had the following metadata:

Table 27: Metadata for PID 1008 dumped using the Procmemdump plugin (SpyEye)

Filename	procmemdump/executable.1008.exe
Size	1,044,480 bytes
SHA1 hash	6c14152bf01e688dcef57a18252d7443595063ee
Fuzzy hash	12288:XHmcoCUyZtwAvAs4wTCyrPT/oHWYUrkf8w0Vnzac1/g/J/vMS:3mfty/wAvN7lrdbkf8w0VnH1/g/J/k

All file metadata was saved for possible future use.

3.3.3.5.2 Virus scanning and file hashing

The SHA1 hash of file *procmemdump/executable.1008.exe* was compared against the SHA1 hashes of the NSRL hash-set but no matches were found.

Fuzzy hash matching of file *procmemdump/executable.1008.exe* was carried out against the carved data files but no matches, even partial, were found. Then fuzzy hash matching was conducted against the *malfind*-dumped, *memdump*-dumped and *procexedump*-dumped files. A 97% match was established between *procexedump/executable.1008.exe* and *procmemdump/executable.1008.exe* indicating they are very similar to one another. This was expected as the only difference between them should be the process's slack space.

Using the six scanners to verify the *procmemdump*-dumped memory sample, ClamAV reported the file to be infected with "Trojan.Backdoor.Bot-1." As none of the other scanners picked up on this, a manual *strings* analysis was warranted.

A vigilant 7, 8, 16 and 32-bit *strings*-based examination of the memory sample failed to find anything suspicious with it. Thus, the detection was most likely a false positive.

3.3.3.5.3 Summary

The *procmemdump* plugin, while having succeeded in dumping the executable code for PID 1008 (*explorer.exe*), did not contain any of the injected code found using the *malfind* plugin.

Finally, the detection of the *procmemdump*-dumped file by ClamAV should be considered a false positive.

3.3.4 Step 4: Examining the registry

The Windows registry serves to complicate and facilitate the investigator's work. It is commonly used by malware to configure system settings for permanent infection. However, the difficulty in working with the registry lies in knowing where to look. The registry is spread out across many data files (commonly known as registry hives) in various locations and each serves a specific purpose with respect to system, application and user configurations. Annex D provides a listing of registry keys commonly used by malware.

3.3.4.1 Hivelist plugin

The purpose of using the *hivelist* plugin is to determine which registry hives[5] are available in the memory image.

Consider the following output from this plugin, using command "*volatility -f spyeye.vmem hivelist*":

Table 28: Volatility output for the Hivelist plugin (SpyEye)

Virtual Address	Physical Address	Filename and Location
0xe22ad700	0x198a4700	\Device\HarddiskVolume1\Documents and Settings\Administrator\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
0xe2239008	0x19413008	\Device\HarddiskVolume1\Documents and Settings\Administrator\NTUSER.DAT
0xe1bd85e0	0x0ec325e0	\Device\HarddiskVolume1\Documents and Settings\LocalService\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
0xe1be0008	0x0ef65008	\Device\HarddiskVolume1\Documents and Settings\LocalService\NTUSER.DAT

[5] A registry hive denotes the actual disk file and its location on disk.

Virtual Address	Physical Address	Filename and Location
0xe1b86400	0x0e684400	\Device\HarddiskVolume1\Documents and Settings\NetworkService\Local Settings\Application Data\Microsoft\Windows\UsrClass.dat
0xe1ba5008	0x0eb3b008	\Device\HarddiskVolume1\Documents and Settings\NetworkService\NTUSER.DAT
0xe1640b60	0x0a6e4b60	\Device\HarddiskVolume1\WINDOWS\system32\config\software
0xe162ab60	0x0a742b60	\Device\HarddiskVolume1\WINDOWS\system32\config\default
0xe16488d0	0x0a76f8d0	\Device\HarddiskVolume1\WINDOWS\system32\config\SECURITY
0xe140fb60	0x04309b60	\Device\HarddiskVolume1\WINDOWS\system32\config\SAM
0xe171db60	0x02e7cb60	[no name]
0xe1035b60	0x02a9eb60	\Device\HarddiskVolume1\WINDOWS\system32\config\system
0xe102e008	0x02a98008	[no name]
0x80670d28	0x00670d28	[no name]

3.3.4.2 Printkey plugin

Using the proposed registry keys identified in Annex D, 952 Volatility *printkey* commands were issued via a script to query the memory image for information pertaining to the various registry hives where this malware may have left traces of its activity. All output was captured and stored in a text file for further analysis.

After running the script, no pertinent information concerning the infection could be found.

3.3.4.3 Userassist plugin

The final registry-based plugin run against the memory image was *userassist*. This plugin has the potential to provide, among other things, registry-based information pertaining to programs run and files opened by the user.

Unfortunately, this plugin did not result in any useful information concerning the infection.

3.3.5 Step 5: Strings analysis

Another technique must be used to extract pertinent information from the memory image concerning the infection. Thus, using the *strings* command it may be possible to obtain additional evidence about the malware and its effect on the underlying computer system.

3.3.5.1 Extraction against plugin-based dumped files

This subsection conducts *strings*-based analysis against only those files successfully obtained using the memory dumping plugins.

3.3.5.1.1 Commands

The following commands were used against the *malfind* and *memdump* plugin-dumped malware samples for PID 1008:

```
$ strings -e s -t d malfind/process.0x823e32f8.0xea000000.dmp | grep -i -P '(keyword1|keyword2|…|keywordn)'

$ strings -e S -t d malfind/process.0x823e32f8.0xea000000.dmp | grep -i -P '(keyword1|keyword2|…|keywordn)'

$ strings -e l -t d malfind/process.0x823e32f8.0xea000000.dmp | grep -i -P '(keyword1|keyword2|…|keywordn)'

$ strings -e L -t d malfind/process.0x823e32f8.0xea000000.dmp | grep -i -P '(keyword1|keyword2|…|keywordn)'

$ strings -e s -t d memdump/1008.dmp | grep -i -P '(keyword1|keyword2|…|keywordn)'

$ strings -e S -t d memdump/1008.dmp | grep -i -P '(keyword1|keyword2|…|keywordn)'

$ strings -e l -t d memdump/1008.dmp | grep -i -P '(keyword1|keyword2|…|keywordn)'

$ strings -e L -t d memdump/1008.dmp | grep -i -P '(keyword1|keyword2|…|keywordn)'
```

These commands carryout case-insensitive (-i) searches using *grep*'s Perl-like (-P) pattern matching; to remove non-pertinent output keyword filters are used. Output can be tuned where *keyword1*, *keyword2*… *keywordn* represent the following word-filters:

- 2b8a408b56eaf3ce0198c91d8a75ec0
- bot_version
- Botnet
- cf.bin
- cleansweep
- config.bin
- cs.exe
- cs.idb

- CurrentVersion\\Policies\\System
- CurrentVersion\\Run
- Dropper
- edc7c152759ba0482bd39db0ea2c4319
- Software\\
- Software\\Microsoft
- Software\\Microsoft\Security Center
- Software\\Microsoft\Windows\CurrentVersion\Policies\System
- Software\Microsoft\Windows\CurrentVersion\Run
- System\\
- System\\ControlSet
- System\\CurrentControlSet
- szmodulefilename
- Trojan

Consider that word-filters *Software*\\, *System*\\ and *CurrentVersion*\\ are indicative of registry hives.

3.3.5.1.2 Pertinent strings

Running the aforementioned commands with the above listed keyword filters resulted in the following pertinent strings, likely applicable to this specific malware:

- *Dropper* : BOT_VERSION = %d, PID = %d, szModuleFileName = "%s"
- *Dropper*!main : CreateMutex->ERROR_ALREADY_EXISTS
- :\cleansweep.exe\cleansweep.exe.Config
- ??\C:\cleansweep.exe\cleansweep.exe.Config
- ??\C:\cleansweep.exe\cleansweep.exe.Manifest
- \??\C:\cleansweep.exe\cleansweep.exe
- \cleansweep.exe
- \cleansweep.exe\cleansweep.exe
- \cleansweep.exe\config.bin
- \Device\HarddiskVolume1\cleansweep.exe
- \Device\HarddiskVolume1\cleansweep.exe\cleansweep.exe
- \Device\HarddiskVolume1\cleansweep.exe\cleansweep.exeN
- \Device\HarddiskVolume1\Documents and Settings\Administrator\Desktop\spyeye\spyeye\cs.exe
- \Documents and Settings\Administrator\Desktop\spyeye\spyeye\cf.bin
- \Documents and Settings\Administrator\Desktop\spyeye\spyeye\config.bin.zip
- \Documents and Settings\Administrator\Desktop\spyeye\spyeye\cs.exe
- \Documents and Settings\Administrator\Desktop\spyeye\spyeye\cs.idb

- \Documents and Settings\Administrator\Desktop\spyeye\spyeye\edc7c152759ba0482bd39db0ea2c4319
- __CLEANSWEEP__
- __CLEANSWEEP_RELOADCFG__
- __CLEANSWEEP_REPALREADYSENDED__
- __CLEANSWEEP_UNINSTALL__
- 2007 Microsoft CleanSweep system
- à½øcleansweep.exe
- bot_version
- C:\cleansweep.exe\cleansweep.exe
- cf.bin
- CF.BIN
- CleanSweep
- cleansweep.exe
- CLEANSWEEP.EXE
- cleansweep.exe
- CLEANSWEEP.EXEr
- CLEANSWEEP_UNINSTALL__
- cleansweepupd.exe
- config.bin
- CONFIG.BIN
- config.bin.zip
- CONFIG.BIN.ZIP
- Content-Disposition: form-data; name="bot_version"
- cs.exe
- CS.EXE
- cs.idb
- CS.IDB
- Description: Microsoft CleanSweep
- edc7c152759ba0482bd39db0ea2c4319
- edc7c152759ba0482bd39db0ea2c4319EDC7C152759BA0482BD39DB0EA2C43192
- Iansweep.exe\cleansweep.exe
- Microsoft CleanSweep
- Norton CleanSweep
- spyeye/cf.binUT
- spyeye/config.bin.zipUT
- spyeye/cs.exeUT
- spyeye/cs.idbUT
- spyeye/edc7c152759ba0482bd39db0ea2c4319UT

3.3.5.1.3 Analysis

Based on the above output, it appears that the malware and its dropper both rely on various configuration files. Moreover, the above output indicates that the dropper verifies if it has been

correctly loaded, creates its mutex and uses false descriptions of itself to fool would-be investigators. Finally, the location of the malware dropper and actual malware, *cleansweep.exe* and *cs.exe*, are now definitively identified.

3.3.5.2 Extraction against memory image

This subsection conducts *strings*-based analysis against the entire memory image file, *spyeye.vmem*.

3.3.5.2.1 Commands

Using the aforementioned keywords the following commands were run against the memory image:

```
$ strings -e s -t d spyeye.vmem | grep -i -P
‘(keyword1|keyword2|…|keywordn)’

$ strings -e S -t d spyeye.vmem | grep -i -P
‘(keyword1|keyword2|…|keywordn)’

$ strings -e l -t d spyeye.vmem | grep -i -P
‘(keyword1|keyword2|…|keywordn)’

$ strings -e L -t d spyeye.vmem | grep -i -P
‘(keyword1|keyword2|…|keywordn)’
```

3.3.5.2.2 Pertinent strings

Applying the aforementioned keywords to the above listed *strings* commands, the following output has been manually pruned for pertinence. Thus, it is possible that some items will have been inadvertently missed. The pertinent output is as follows:

- *Dropper* : BOT_VERSION = %d, PID = %d, szModuleFileName = "%s"
- *Dropper*!main : CreateMutex->ERROR_ALREADY_EXISTS
- . DropPercent %d
- :\cleansweep.exe\cleansweep.e
- ; File Name : \\.host\Shared Folders\rolson\malshare\families\spyeye\cs.exe
- \\.host\Shared Folders\rolson\malshare\families\spyeye\cs.exe
- \cleansweep.exe
- \cleansweep.exe\cleansweep.exe
- \cleansweep.exe\config.bin
- \Device\HarddiskVolume1\cleansweep.exe
- \Device\HarddiskVolume1\cleansweep.exe\cleansweep.exe
- \Device\HarddiskVolume1\cleansweep.exe\cleansweep.exeN

- \Device\HarddiskVolume1\Documents and Settings\Administrator\Desktop\spyeye\spyeye\cs.exe
- \Documents and Settings\Administrator\Desktop\spyeye\spyeye\cf.bin
- \Documents and Settings\Administrator\Desktop\spyeye\spyeye\config.bin.zip
- \Documents and Settings\Administrator\Desktop\spyeye\spyeye\cs.exe
- \Documents and Settings\Administrator\Desktop\spyeye\spyeye\cs.idb
- \Documents and Settings\Administrator\Desktop\spyeye\spyeye\edc7c152759ba0482bd39db0ea2c4319
- __CLEANSWEEP
- __CLEANSWEEP__
- __CLEANSWEEP_RELOADCFG__
- __CLEANSWEEP_REPALREADYSENDED__
- __CLEANSWEEP_UNINSTALL__
- _cleansweep_0
- à½øcleansweep.exe
- cleansweep.exe
- CLEANSWEEP.EXE
- CLEANSWEEP.EXEr
- CLEANSWEEP_UNINSTALL__
- cleansweepupd.exe
- config.bin
- CONFIG.BIN
- config.bin.zip
- CONFIG.BIN.ZIP
- Content-Disposition: form-data; name="bot_version"
- cs.exe
- CS.EXE
- cs.idb
- CS.IDB
- -data; name="bot_version"
- Dc:\documents and settings\administrator\desktop\spyeye\spyeye\cs.exe
- Description: Microsoft CleanSweep
- Dropping reinit. DropPercent %d
- edc7c152759ba0482bd39db0ea2c4319
- edc7c152759ba0482bd39db0ea2c4319EDC7C152759BA0482BD39DB0EA2C43192
- edc7c152759ba0482bd39db0ea2c4319UT
- Iansweep.exe\cleansweep.exe
- MD5: edc7c152759ba0482bd39db0ea2c4319 added to sample package

- m-data; name="bot_version"
- Microsoft CleanSweep
- Norton CleanSweep
- scription: Microsoft CleanSweep
- spyeye/cf.binUT
- spyeye/config.bin.zipUT
- spyeye/cs.exeUT
- spyeye/cs.idbUT
- spyeye/edc7c152759ba0482bd39db0ea2c4319UT

3.3.5.2.3 Analysis

The above output is similar to that obtained from the malware (see Section 3.3.5.1 for details). Although additional context would certainly help, it appears that the malware and its dropper both rely on various configuration files. Moreover, the above output indicates that the dropper verifies if it has been correctly loaded, creates its mutex and uses false descriptions of itself to fool would-be investigators. Finally, the location of the malware dropper and actual malware, *cleansweep.exe* and *cs.exe*, are now definitively identified.

4 Conclusion

It can be concluded that using sound investigative footwork, combined with the capabilities of the Volatility memory analysis framework, investigators can readily analyse and investigate suspected memory-based infections.

The Prolaco worm and SpyEye Trojan horse examined herein were not of equal difficulty. Prolaco was simpler to detect than SpyEye, as it made no effort to hide itself through code injection, although it was not detectable using a standard process listing (i.e. *pslist*). On the other hand, SpyEye uses code injection to hide itself.

Unlike the Zeus analysis, this work relied far less on virus and malware reports and alerts, although pertinent documentation was made for the reader.

What these two analyses have emphasized is the fallacy of overreliance on any one tool or technique, be it malware scanners, memory frameworks or *strings* analysis. Instead, these tools and techniques may be used to maximize an investigator's ability to analyse a suspected memory image efficiently. It was evident from the two analyses that excessive confidence of scanner-based results can lead an investigator astray. To counterbalance this, the investigator should not rely on any single scanner result; instead, credence should be emphasised when multiple scanners agree on a file (or memory sample) being possibly infected. Moreover, when necessary, the investigator should not hesitate from using *strings*-based analysis looking in order to look for common malware indicators.

Throughout this document, as based on the proposed and lightly amended methodology, the author has demonstrated the manner in which a forensic memory analysis can be conducted by non-memory specialists. Thus, even novice memory investigators can successfully conduct difficult memory analyses, when armed with straightforward tools, techniques and methodology, as well as some basic background concerning the suspected infection.

However, not all analyses can rely on many well-prepared virus reports, as the analysis of the Prolaco worm clearly demonstrated. Furthermore, not all investigations will be carried out against known malware as the threat is constantly evolving. Nevertheless, the techniques and methodology presented herein will be of use, to varying extents, against these newer malware.

This document, the second in a series of many, has guided the reader through two well-known memory infections with the expectation of building a sufficient compendium of knowledge for memory analysis for use by novice and expert memory analysts alike. While the degree of difficulty varies substantially from case to case, the Volatility framework, when combined with investigative knowhow, tools, techniques and methodology is a highly adept analysis-based framework.

References

[1] Kiguolis, Ugnius. Alg.exe information? What is alg.exe. Informational web site. 2Spyware.com. Unknown date. http://www.2-spyware.com/file-alg-exe.html.

[2] Sophos. W32/Prolaco-F. informational web site. Sophos.com. July 2010. https://secure2.sophos.com/en-us/threat-center/threat-analyses/viruses-and-spyware/W32~Prolaco-F/detailed-analysis.aspx.

[3] SecurityHome.eu. Worm: Win32/Prolaco.BC. Informational web report. SecuirityHome.eu. June 2010. http://www.securityhome.eu/malware/malware_pdf.php?mal_id=9165136514c1a3a9f832010.00916478.

[4] Avira.com. Full description: Worm/Prolaco.C.2. Informational web site. July 2010. Avira.com. http://www.avira.com/en/support-threats-description/tid/5377/tlang/en.

[5] McAfee.com. Virus Profile: W32/STD.worm.gen! 31C4A278EDB0. Informational web site. McAfee.com. Unknown date. http://home.mcafee.com/virusinfo/virusprofile.aspx?key=1033122#.

[6] Sood, Aditya K. SpyEye Banking Trojan. SecNiche Security. Presentation at ToorCon 12 – San Diego 2010. 2010. http://www.secniche.org/presentations/toorcon_sandiego_2010_adityaks.pdf.

[7] Sood, Aditya K. and Enbody, Richard J. Spying on SpyEye: What Lies Beneath? SecNiche Security and Department of Computer Science and Engineering (Michigan State University). Presentation at HackInTheBox Security Conference, Amsterdam. 2011. http://www.secniche.org/presentations/hitb_ams_2011_adityaks.pdf.

[8] McRee, Russ. Memory Analysis wuth DumpIt and Volatility. Journal article. ISSA Journal. September 2011. http://holisticinfosec.org/toolsmith/pdf/september2011.pdf.

[9] Bodmer, Sean. SpyEye being kicked to the curb by its customers? Research Note. Damballa.com. 2012. https://www.damballa.com/downloads/r_pubs/RN_SpyEye-Kicked-to-Curb_Bodmer.pdf.

[10] Heriyanto, Andri P. What is the Proper Forensics Approach on Trojan Banking Malware Incidents? Research paper. Edith Cowan University. 2012. http://ro.ecu.edu.au/cgi/viewcontent.cgi?article=1106&context=adf&sei-redir=1&referer=http%3A%2F%2Fwww.google.ca%2Furl%3Fsa%3Dt%26rct%3Dj%26q%3DWHAT%2520IS%2520THE%2520PROPER%2520FORENSICS%2520APPROACH%2520ON%2520TROJAN%2520BANKING%2520MALWARE%2520INCIDENTS%253F%26source%3Dweb%26cd%3D1%26cad%3Drja%26ved%3D0CCoQFjAA%26url%3Dhttp%253A%252F%252Fro.ecu.edu.au%252Fcgi%252Fviewcontent.cgi%253Farticle%253D1106%2526context%253Dadf%26ei%3DR9oLUuKFNMP7yAHCroDYDg%26usg%3DAFQjCNGgy2eEtW1KPvWbLsns3rGlidtwDw#search=%22WHAT%20PROPER%20FORENSICS%20APPROACH%20TROJAN%20BANKING%20MALWARE%20INCIDENTS%3F%22.

[11] Nayyar, Harshit. Clash of the Titans: ZeuS v SpyEye. SANS GIAC Gold Certification report. SANS. 2010. http://www.sans.org/reading-room/whitepapers/malicious/clash-titans-zeus-SpyEye-33393.

[12] Serban, Liviu. TR/Spy.Spyeye Analysis. Technical paper. Avira GmbH. Unknown date. http://techblog.avira.com/wp-content/uploads/2011/03/Analysis-of-TR.Spy_.SpyEye.pdf.

[13] Kharouni, Loucif; Stevens, Kevin, et al. From Russia to Hollywood: Turning The Tables On A Spyeye Cybercrime Ring. Research paper. Trend Micro. 2011. http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp_turning-the-tables_SpyEye-cibercrime-ring.pdf.

[14] Kharouni, Loucif. Automating Online Banking Fraud. Automatic Transfer System: The Latest Cybercrime Toolkit Feature. Research paper. Trend Micro. 2012. http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp_automating_online_banking_fraud.pdf.

[15] Kirk, Jeremy. SpyEye Trojan defeating online banking defenses. Online article. Computerworld.net. July 2011. http://www.computerworld.com/s/article/9218645/SpyEye_Trojan_defeating_online_banking_defenses.

[16] Waugh, Rob. New PC virus doesn't just steal your money - it creates fake online bank statements so you don't know it's gone. Online article. Dailymail.co.uk. January 2012. http://www.dailymail.co.uk/sciencetech/article-2083271/SpyEye-trojan-horse-New-PC-virus-steals-money-creates-fake-online-bank-statements.html.

[17] Mieres, Jorge. SpyEye Bot: Analysis of a new alternative scenario crimeware. Technical paper. Malware Intelligence. February 2010. http://www.malwareint.com/docs/SpyEye-analysis-en.pdf.

[18] Mieres, Jorge. SpyEye Bot (Part two): Conversations with the creator of crimeware. Technical paper. Malware Intelligence. February 2010. http://www.malwareint.com/docs/SpyEye-analysis-ii-en.pdf.

[19] Volatility. CommandReference: Example usage cases and output for Volatility 2.0 commands. Online command reference. Volatility. February 2012. http://code.google.com/p/volatility/wiki/CommandReference.

[20] Vatamanu, Chris. Win32.Worm.Prolaco.S. Online technical article. BitDefender. Unknown date. http://www.bitdefender.com/VIRUS-1000638-en--Win32-Worm-Prolaco-S.html.

[21] Wikipedia. Client/Server Runtime Subsystem. Online encyclopaedic entry. Wikimedia Foundation Inc. Aprl 2013. http://en.wikipedia.org/wiki/Client/Server_Runtime_Subsystem.

[22] Carbone, Richard. Malware memory analysis for non-specialists: Investigating a publicly available memory image of the Zeus Trojan horse. Technical Memorandum. Defence R&D Canada – Valcartier. TM 2013-018. April 2013.

[23] Carbone, Richard. File recovery and data extraction using automated data recovery tools: A balanced approach using Windows and Linux when working with an unknown disk image and filesystem. Technical memorandum. TM 2009-161. Defence R&D Canada - Valcartier. January 2013. http://cradpdf.drdc-rddc.gc.ca/PDFS/unc122/p531895_A1b.pdf.

This page intentionally left blank.

Annex A Anti-virus scanner logs for carved data files

A.1 Prolaco

In all, two virus matches were identified between the various scanners. The matches have been identified in the following output.

A.1.1 Avast

The Avast anti-virus scanner was unable to detect any malware whatsoever for the recovered data files.

A.1.2 AVG

./recup_dir.2/f0139704.exe Virus identified Worm/Generic2.FQ ← **Match 1**
./recup_dir.2/f0233072.dll Virus found Win32/Heur
./recup_dir.2/f0235672.dll Virus found Win32/Heur ← **Match 2**
./recup_dir.2/f0236256.dll Virus found Win32/Heur

A.1.3 BitDefender

./recup_dir.2/f0153976.exe infected: Gen:Variant.Renos.14

A.1.4 ClamAV

./recup_dir.1/f0008976.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.1/f0014888.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.1/f0016968.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.1/f0016992.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.1/f0021024.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.1/f0023984.dll: PUA.Win32.Packer.Msvcpp FOUND
./recup_dir.1/f0030600.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.1/f0053120.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.1/f0055560.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.1/f0062272.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.1/f0084616.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.1/f0089144.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.1/f0097936.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.1/f0102264.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.1/f0102704.exe: PUA.Win32.Packer.BorlandCpp-8 FOUND
./recup_dir.1/f0103800.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.1/f0109736.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.1/f0115072.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND

./recup_dir.1/f0123048.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.1/f0129560.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.1/f0131920.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.2/f0139704.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND ← **Match 1**
./recup_dir.2/f0139720.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.2/f0142536.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.2/f0152256.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.2/f0156760.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.2/f0160720.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.2/f0161656.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.2/f0162968.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.2/f0167040.dll: PUA.Win32.Packer.Msvcpp FOUND
./recup_dir.2/f0167072.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.2/f0167104.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.2/f0167120.dll: PUA.Win32.Packer.Msvcpp FOUND
./recup_dir.2/f0167168.exe: PUA.Win32.Packer.Msvcpp FOUND
./recup_dir.2/f0167280.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.2/f0167288.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.2/f0167344.dll: PUA.Win32.Packer.Msvcpp FOUND
./recup_dir.2/f0167352.dll: PUA.Win32.Packer.Msvcpp FOUND
./recup_dir.2/f0167680.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.2/f0167960.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.2/f0167968.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.2/f0167992.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.2/f0168256.dll: PUA.Win32.Packer.BorlandDelphi-2 FOUND
./recup_dir.2/f0169264.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.2/f0185552.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.2/f0194104.dll: PUA.Win32.Packer.Msvcpp FOUND
./recup_dir.2/f0204752.dll: PUA.Win32.Packer.Msvcpp FOUND
./recup_dir.2/f0207224.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.2/f0216392.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.2/f0217936.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.2/f0219024.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.2/f0219512.exe: PUA.Win32.Packer.Msvcpp FOUND
./recup_dir.2/f0228584.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.2/f0229384.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.2/f0233040.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.2/f0234168.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.2/f0235672.dll: PUA.Win32.Packer.Msvcpp FOUND ← **Match 2**
./recup_dir.2/f0237232.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.2/f0244384.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.2/f0245136.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
./recup_dir.2/f0245336.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND

A.1.5 F-Prot

The FRISK F-Prot anti-virus scanner was unable to detect any malware whatsoever for the recovered data files.

A.1.6 McAfee

The McAfee anti-virus scanner was unable to detect any malware whatsoever for the recovered data files.

A.2 SpyEye

In all, five virus matches were identified between the various scanners. The matches have been identified in the following output.

A.2.1 Avast

The Avast anti-virus scanner was unable to detect any malware whatsoever for the recovered data files.

A.2.2 AVG

recup_dir.1/f0022784.dll Virus found Win32/Heur
recup_dir.1/f0167784.dll Virus found Win32/Heur
recup_dir.1/f0176416.dll Virus found Win32/Heur
recup_dir.1/f0184448.dll Virus found Win32/Heur
recup_dir.1/f0227256.dll Virus found Win32/Heur
recup_dir.1/f0242368.exe Virus found Win32/Heur
recup_dir.1/f0263584.exe Virus found Win32/Heur ← **Match 1**
recup_dir.1/f0264744.dll Virus found Win32/Heur
recup_dir.2/f0263296.dll Virus found Win32/Heur ← **Match 2**
recup_dir.2/f0263528.dll Virus found Win32/Heur
recup_dir.2/f0305128.dll Virus found Win32/Heur ← **Match 3**
recup_dir.2/f0335944.dll Virus found Win32/Heur
recup_dir.2/f0465128.dll Virus found Win32/Heur
recup_dir.2/f0513352.dll Virus found Win32/Heur
recup_dir.2/f0514448.exe Virus found Win32/Heur
recup_dir.2/f0515800.dll Virus found Win32/Heur
recup_dir.2/f0612712.dll Virus found Win32/Heur
recup_dir.2/f0631784.dll Virus found Win32/Heur
recup_dir.2/f0633168.dll Virus found Win32/Heur
recup_dir.2/f0702456.dll Virus found Win32/Heur
recup_dir.3/f0757168.dll Virus found Win32/Heur
recup_dir.3/f0767272.dll Virus found Win32/Heur
recup_dir.3/f0769864.dll Virus found Win32/Heur
recup_dir.3/f0952760.dll Virus found Win32/Heur ← **Match 4**
recup_dir.3/f0993000.dll Virus found Win32/Heur

A.2.3 BitDefender

recup_dir.1/f0292960.exe infected: Gen:Variant.Kazy.9508
recup_dir.2/f0612984.exe infected: Gen:Trojan.Heur.JP.rmW@a4vC9Ke
recup_dir.2/f0630512.exe infected: Gen:Trojan.Heur.rmW@!Fh9mcb ← **Match 5**

A.2.4 ClamAV

recup_dir.1/f0020448.dll: PUA.Win32.Packer.Msvcpp FOUND
recup_dir.1/f0061184.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.1/f0103960.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.1/f0158928.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.1/f0182456.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.1/f0184080.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.1/f0186304_netmsg.DLL: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.1/f0186616.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.1/f0235160.dll: PUA.Win32.Packer.SetupExeSection FOUND
recup_dir.1/f0258912.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.1/f0263576.exe: PUA.Win32.Packer.Msvcpp FOUND
recup_dir.1/f0263584.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND ← **Match 1**
recup_dir.1/f0263600.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.1/f0263632.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.1/f0263640.dll: PUA.Win32.Packer.Msvcpp FOUND
recup_dir.1/f0263744.exe: PUA.Win32.Packer.Msvcpp FOUND
recup_dir.1/f0263960.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.1/f0267776.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.1/f0269040.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.1/f0280160.dll: PUA.Win32.Packer.Msvcpp FOUND
recup_dir.1/f0285144.dll: PUA.Win32.Packer.Msvcpp FOUND
recup_dir.1/f0295536.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.1/f0297488.dll: PUA.Win32.Packer.Msvcpp FOUND
recup_dir.2/f0263296.dll: PUA.Win32.Packer.Msvcpp FOUND ← **Match 2**
recup_dir.2/f0263560.exe: PUA.Win32.Packer.SetupExeSection FOUND
recup_dir.2/f0263568.dll: PUA.Win32.Packer.Msvcpp FOUND
recup_dir.2/f0305128.dll: PUA.Win32.Packer.Msvcpp FOUND ← **Match 3**
recup_dir.2/f0313568.dll: PUA.Win32.Packer.Msvcpp FOUND
recup_dir.2/f0322664.dll: PUA.Win32.Packer.Pequake-3 FOUND
recup_dir.2/f0324792.dll: PUA.Win32.Packer.Msvcpp FOUND
recup_dir.2/f0342664.dll: PUA.Win32.Packer.Upx-28 FOUND
recup_dir.2/f0442416.dll: PUA.Win32.Packer.BorlandDelphiKo FOUND
recup_dir.2/f0444800.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.2/f0445384.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.2/f0452400.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.2/f0463224.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.2/f0463904.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.2/f0464272.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.2/f0465776.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.2/f0466144.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.2/f0467744.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.2/f0469912.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.2/f0472848.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.2/f0484336.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.2/f0487192.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.2/f0492176.dll: PUA.Win32.Packer.Msvcpp FOUND

recup_dir.2/f0497368.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.2/f0500840.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.2/f0501472.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.2/f0501816.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.2/f0516456.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.2/f0518416.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.2/f0525760.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.2/f0534888.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.2/f0558272.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.2/f0598232.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.2/f0614640.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.2/f0615928.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.2/f0617048.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.2/f0623960.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.2/f0628440.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.2/f0630512.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND ← **Match 5**
recup_dir.2/f0636264.dll: PUA.Win32.Packer.Upx-28 FOUND
recup_dir.2/f0647968.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.2/f0652416.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.2/f0653032.dll: PUA.Win32.Packer.Msvcpp FOUND
recup_dir.2/f0655848.dll: PUA.Win32.Packer.BorlandDelphiKo FOUND
recup_dir.2/f0659656.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.2/f0660216.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.2/f0676176.dll: PUA.Win32.Packer.Upx-28 FOUND
recup_dir.2/f0680448.dll: PUA.Win32.Packer.Msvcpp FOUND
recup_dir.2/f0701040.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.2/f0705216.dll: PUA.Win32.Packer.BorlandDelphiKo FOUND
recup_dir.3/f0714224.dll: PUA.Win32.Packer.Msvcpp FOUND
recup_dir.3/f0729064.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0751616.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0764912.dll: PUA.Win32.Packer.BorlandDelphiKo FOUND
recup_dir.3/f0781232.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0790312.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0802232.dll: PUA.Win32.Packer.SetupExeSection FOUND
recup_dir.3/f0807104.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0811248.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0811440.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0815704.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0818848.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0824864.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0826160.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0826616.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0844136.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0848160.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0855568.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0859976.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0864056.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0878336.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND

recup_dir.3/f0893272.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0903456.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0908560.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0909344.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0910464.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0910920.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0913768.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0921696.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0937960.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0942424.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0943056.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0946768.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0950288.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0952760.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND ← **Match 4**
recup_dir.3/f0953808.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0958296.dll: PUA.Win32.Packer.BorlandDelphiKo FOUND
recup_dir.3/f0963552.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0966440.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0979144.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0980008.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0981976.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0987856.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0990424.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f0990816.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f1001008.exe: PUA.Win32.Packer.MsVisualCpp-2 FOUND
recup_dir.3/f1046000.dll: PUA.Win32.Packer.MsVisualCpp-2 FOUND

A.2.5 F-Prot

recup_dir.1/f0111368.exe <W32/Heuristic-CO3!Eldorado (not disinfectable)>
recup_dir.2/f0306064.dll <W32/Heuristic-COC!Eldorado (not disinfectable)>
recup_dir.2/f0424400.dll <W32/Heuristic-COC!Eldorado (not disinfectable)>
recup_dir.2/f0554088.exe <W32/Heuristic-CO3!Eldorado (not disinfectable)>

A.2.6 McAfee

recup_dir.2/f0542616.exe ... Found the Downloader-ASH.gen.g trojan !!!

This page intentionally left blank.

Annex B Volatility Windows-based plugins

The following is a complete list of the default Windows-based plugins provided by Volatility version 2.2:

Table B.1: List of Volatility 2.2 plugins.

Plugin	**Capability (as per *Volatility --help* output)**
apihooks	Detect API hooks in process and kernel memory
atoms	Print session and window station atom tables
atomscan	Pool scanner for _RTL_ATOM_TABLE
bioskbd	Reads the keyboard buffer from Real Mode memory
callbacks	Print system-wide notification routines
clipboard	Extract the contents of the windows clipboard
cmdscan	Extract command history by scanning for _COMMAND_HISTORY
connections	Print list of open connections [Windows XP and 2003 Only]
connscan	Scan Physical memory for _TCPT_OBJECT objects (tcp connections)
consoles	Extract command history by scanning for _CONSOLE_INFORMATION
crashinfo	Dump crash-dump information
deskscan	Poolscaner for tagDESKTOP (desktops)
devicetree	Show device tree
dlldump	Dump DLLs from a process address space
dlllist	Print list of loaded dlls for each process
driverirp	Driver IRP hook detection
driverscan	Scan for driver objects _DRIVER_OBJECT
envars	Display process environment variables
eventhooks	Print details on windows event hooks
evtlogs	Extract Windows Event Logs (XP/2003 only)
filescan	Scan Physical memory for _FILE_OBJECT pool allocations
gahti	Dump the USER handle type information

Plugin	**Capability (as per *Volatility --help* output)**
gditimers	Print installed GDI timers and callbacks
gdt	Display Global Descriptor Table
getservicesids	Get the names of services in the Registry and return Calculated SID
getsids	Print the SIDs owning each process
handles	Print list of open handles for each process
hashdump	Dumps passwords hashes (LM/NTLM) from memory
hibinfo	Dump hibernation file information
hivedump	Prints out a hive
hivelist	Print list of registry hives.
hivescan	Scan Physical memory for _CMHIVE objects (registry hives)
idt	Display Interrupt Descriptor Table
imagecopy	Copies a physical address space out as a raw DD image
imageinfo	Identify information for the image
impscan	Scan for calls to imported functions
kdbgscan	Search for and dump potential KDBG values
kpcrscan	Search for and dump potential KPCR values
ldrmodules	Detect unlinked DLLs
lsadump	Dump (decrypted) LSA secrets from the registry
malfind	Find hidden and injected code
memdump	Dump the addressable memory for a process
memmap	Print the memory map
messagehooks	List desktop and thread window message hooks
moddump	Dump a kernel driver to an executable file sample
modscan	Scan Physical memory for _LDR_DATA_TABLE_ENTRY objects
modules	Print list of loaded modules
mutantscan	Scan for mutant objects _KMUTANT
patcher	Patches memory based on page scans

Plugin	**Capability (as per *Volatility --help* output)**
printkey	Print a registry key, and its subkeys and values
procexedump	Dump a process to an executable file sample
procmemdump	Dump a process to an executable memory sample
pslist	Print all running processes by following the EPROCESS lists
psscan	Scan Physical memory for _EPROCESS pool allocations
pstree	Print process list as a tree
psxview	Find hidden processes with various process listings
raw2dmp	Converts a physical memory sample to a windbg crash dump
screenshot	Save a pseudo-screenshot based on GDI windows
sessions	List details on _MM_SESSION_SPACE (user logon sessions)
shimcache	Parses the Application Compatibility Shim Cache registry key
sockets	Print list of open sockets
sockscan	Scan Physical memory for _ADDRESS_OBJECT objects (tcp sockets)
ssdt	Display SSDT entries
strings	Match physical offsets to virtual addresses (may take a while, VERY verbose)
svcscan	Scan for Windows services
symlinkscan	Scan for symbolic link objects
thrdscan	Scan physical memory for _ETHREAD objects
threads	Investigate _ETHREAD and _KTHREADs
timers	Print kernel timers and associated module DPCs
userassist	Print userassist registry keys and information
userhandles	Dump the USER handle tables
vaddump	Dumps out the vad sections to a file
vadinfo	Dump the VAD info
vadtree	Walk the VAD tree and display in tree format
vadwalk	Walk the VAD tree
volshell	Shell in the memory image

Plugin	**Capability (as per *Volatility --help* output)**
windows	Print Desktop Windows (verbose details)
wintree	Print Z-Order Desktop Windows Tree
wndscan	Pool scanner for tagWINDOWSTATION (window stations)
yarascan	Scan process or kernel memory with Yara signatures

Annex C NSRL file hash matches for carved data files

C.1 Prolaco

This annex provides a listing of those carved data files obtained in Section 2.2.3 that matched the SHA1 hashes of the NSRL hash-set (March 2013). In all, two unique NSRL-based SHA1 matches were obtained.

These unique SHA1-filename matches are as follows:

Table C.1: SHA1 hash vs. NSRL filename for carved data files (Prolaco)

SHA1 hash	Filename
9537335B7EDA9AE3D1C125BE7BAC3161D5B853B8	comctl.man
9537335B7EDA9AE3D1C125BE7BAC3161D5B853B8	COMCTL.MAN
9537335B7EDA9AE3D1C125BE7BAC3161D5B853B8	X86_POLICY.6.0.MICROSOFT.WINDOWS.COMMON-CONTROLS_6595B64144CCF1DF_6.0.2600.2180_X-WW_EB84B25E.MANIFEST
C5B52B71F4C5F933815D7D606175EA0BB37DC548	controls.man
C5B52B71F4C5F933815D7D606175EA0BB37DC548	CONTROLS.MAN
C5B52B71F4C5F933815D7D606175EA0BB37DC548	X86_MICROSOFT.WINDOWS.COMMON-CONTROLS_6595B64144CCF1DF_6.0.2600.2180_X-WW_A84F1FF9.MANIFEST

C.2 SpyEye

This annex provides a listing of those carved data files obtained in Section 3.2.3 that matched the SHA1 hashes of the NSRL hash-set (March 2013). In all, six unique NSRL-based SHA1 matches were obtained. In turn, these six hashes matched up against twelve unique filenames.

These unique SHA1-filename matches are as follows:

Table C.2: SHA1 hash vs. NSRL filename for carved data files (SpyEye)

SHA1 hash	Filename
15740B197555BA8E162C37A60BA655151E3BEBAE	index.dat
67DE4E3707D69562F8D57558E6CC5144274D96AD	__0X002C

SHA1 hash	Filename
6F9F663CDFBC2592EAB4C43FEE359EFFD37D60F2	dxgthk.sys
6F9F663CDFBC2592EAB4C43FEE359EFFD37D60F2	DXGTHK.SYS
80EB8A76E5579B0136281E4DD4E2D4E56B249E4C	null.sys
80EB8A76E5579B0136281E4DD4E2D4E56B249E4C	NULL.SYS
E07EE000BC06B455534D8A517305C1208D30306B	audstub.sys
FB33FD00711440B9D0F3B3D526A753ED75640797	Windows Navigations Start.wav
FB33FD00711440B9D0F3B3D526A753ED75640797	navstart.wav
FB33FD00711440B9D0F3B3D526A753ED75640797	xpstart.wa!
FB33FD00711440B9D0F3B3D526A753ED75640797	xpstart.wav
FB33FD00711440B9D0F3B3D526A753ED75640797	XPStart.wav

Annex D Commonly used registry keys in a typical malware infection

D.1 Recommended registry keys for use with Volatility

Based on the author's own use and research of various Windows registry keys commonly used by malware, the following keys are recommended for evaluation. These keys are readily integrated into scripts using appropriate Volatility-based *printkey* plugin commands.

However, these keys will not work against all versions of Windows. Some apply to 2000/XP systems while others apply to recent versions of Windows. The reader's success in using these keys will undoubtedly vary according to the underlying Windows platform to be analysed and the malware's propensity for using the registry.

The proposed keys have been aggregated and their preceding HKLM\Software, HKLM\System, HKCU\Software and HKCU based information stripped so that they can be readily used by Volatility.

The following keys are used in this work against both the Prolaco worm and SpyEye Trojan horse:

- Classes\Local Settings\Software\Microsoft\Windows\Shell\MuiCache
- Control Panel\Desktop
- Control Panel\Desktop\ScreenSaveActive
- ControlSet001\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile\AuthorizedApplications\List
- CurrentControlSet\Control\Session Manager\AppCertDlls
- CurrentControlSet\Control\Session Manager\AppCompatCache\AppCompatCache
- CurrentControlSet\Control\Session Manager\AppCompatibility\AppCompatCache
- CurrentControlSet\Control\SessionManager\Memory Management
- CurrentControlSet\Services
- Microsoft\Active Setup\Installed Components
- Microsoft\DirectPlugin
- Microsoft\Internet Explorer\CustomizeSearch
- Microsoft\Internet Explorer\Main
- Microsoft\Internet Explorer\Main\Default_Page_URL
- Microsoft\Internet Explorer\Main\Default_Search_URL
- Microsoft\Internet Explorer\Main\HomeOldSP
- Microsoft\Internet Explorer\Main\Local Page
- Microsoft\Internet Explorer\Main\Search Bar
- Microsoft\Internet Explorer\Main\Search Page
- Microsoft\Internet Explorer\Main\SearchAssistant
- Microsoft\Internet Explorer\Main\SearchURL
- Microsoft\Internet Explorer\Main\Start Page

- Microsoft\Internet Explorer\Main\Use Search Asst
- Microsoft\Internet Explorer\Search
- Microsoft\Internet Explorer\Search Bar
- Microsoft\Internet Explorer\Search\CustomizeSearch
- Microsoft\Internet Explorer\Search\SearchAssistant
- Microsoft\Internet Explorer\SearchURL
- Microsoft\Internet Explorer\Toolbar
- Microsoft\Internet Explorer\TypedURLs
- Microsoft\Windows NT\CurrentVersion\Terminal Server\Install\Software\Microsoft\Windows\CurrentVersion\Run
- Microsoft\Windows NT\CurrentVersion\Terminal Server\Install\Software\Microsoft\Windows\CurrentVersion\Runonce
- Microsoft\Windows NT\CurrentVersion\Terminal Server\Install\Software\Microsoft\Windows\CurrentVersion\RunonceEx
- Microsoft\Windows NT\CurrentVersion\Windows
- Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs
- Microsoft\Windows NT\CurrentVersion\Windows\Load
- Microsoft\Windows NT\CurrentVersion\Winlogon
- Microsoft\Windows NT\CurrentVersion\Winlogon\Notify
- Microsoft\Windows NT\winlogon\userinit
- Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects
- Microsoft\Windows\CurrentVersion\Explorer\ComDlg32\LastVisitedMRU
- Microsoft\Windows\CurrentVersion\Explorer\ComDlg32\OpenSaveMRU
- Microsoft\Windows\CurrentVersion\Explorer\RecentDocs
- Microsoft\Windows\CurrentVersion\Explorer\RunMRU
- Microsoft\Windows\CurrentVersion\Explorer\SharedTaskScheduler
- Microsoft\Windows\CurrentVersion\Explorer\ShellExecuteHooks
- Microsoft\Windows\CurrentVersion\Explorer\UserAssist
- Microsoft\Windows\CurrentVersion\Internet Settings
- Microsoft\Windows\CurrentVersion\Internet Settings\EnableAutodial
- Microsoft\Windows\CurrentVersion\Internet Settings\EnableHttp1_1
- Microsoft\Windows\CurrentVersion\Internet Settings\MaxConnectionsPer1_0Server
- Microsoft\Windows\CurrentVersion\Internet Settings\MaxConnectionsPerServer
- Microsoft\Windows\CurrentVersion\Internet Settings\ProxyEnable
- Microsoft\Windows\CurrentVersion\Internet Settings\ProxyHttp1.1
- Microsoft\Windows\CurrentVersion\Internet Settings\ProxyOverride
- Microsoft\Windows\CurrentVersion\Internet Settings\ProxyServer
- Microsoft\Windows\CurrentVersion\Policies\Explorer\Run
- Microsoft\Windows\CurrentVersion\Run
- Microsoft\Windows\CurrentVersion\RunOnce
- Microsoft\Windows\CurrentVersion\RunOnce\Setup
- Microsoft\Windows\CurrentVersion\RunOnceEx
- Microsoft\Windows\CurrentVersion\RunServices

- Microsoft\Windows\CurrentVersion\RunServicesOnce
- Microsoft\Windows\CurrentVersion\ShellServiceObjectDelayLoad
- Microsoft\Windows\CurrentVersion\URL
- Microsoft\Windows\CurrentVersion\URL\DefaultPrefix
- Microsoft\Windows\CurrentVersion\URL\Prefixes
- Microsoft\Windows\ShellNoRoam\MUICache

D.2 Printkey-based script

The aforementioned keys can be readily integrated into scripts. For example, consider the following Volatility *printkey* command:

> $ volatility -f spyeye.vmem printkey -o 0xe1c49008 -K ″Microsoft\Windows\CurrentVersion\RunServices″

D.3 Root Registry Keys

The author-proposed registry keys are based on the following root registry keys:

> HKEY_CURRENT_USER
>
> HKEY_CURRENT_USER\Software
>
> HKEY_LOCAL_MACHINE\Software
>
> HKEY_LOCAL_MACHINE\System

This page intentionally left blank.

Annex E Fuzzy hashes for Malfind plugin dumped processes

This annex lists the fuzzy hash matches for those memory sample files dumped using the *malfind* plugin from the SpyEye memory image. They are listed in a match-based descending order as follows:

Table E.1: Fuzzy hashes for Malfind-dumped processes (SpyEye)

Filename 1	Filename 2	Match (in %)
process.0x81f4b020.0xea50000.dmp	process.0x81ebd300.0xea50000.dmp	(100)
process.0x81f5e020.0xea50000.dmp	process.0x81ec2020.0xea50000.dmp	(100)
process.0x82067858.0xea50000.dmp	process.0x81ec2020.0xea50000.dmp	(100)
process.0x82067858.0xea50000.dmp	process.0x81f5e020.0xea50000.dmp	(100)
process.0x8207d5f0.0xea50000.dmp	process.0x81ebd300.0xea50000.dmp	(100)
process.0x8207d5f0.0xea50000.dmp	process.0x81f4b020.0xea50000.dmp	(100)
process.0x820bd760.0x15d0000.dmp	process.0x820bd760.0x13d0000.dmp	(100)
process.0x820bd760.0x19d0000.dmp	process.0x820bd760.0x13d0000.dmp	(100)
process.0x820bd760.0x19d0000.dmp	process.0x820bd760.0x15d0000.dmp	(100)
process.0x820bd760.0x1bd0000.dmp	process.0x820bd760.0x13d0000.dmp	(100)
process.0x820bd760.0x1bd0000.dmp	process.0x820bd760.0x15d0000.dmp	(100)
process.0x820bd760.0x1bd0000.dmp	process.0x820bd760.0x19d0000.dmp	(100)
process.0x8214ba18.0xea50000.dmp	process.0x81ec2020.0xea50000.dmp	(100)
process.0x8214ba18.0xea50000.dmp	process.0x81f5e020.0xea50000.dmp	(100)
process.0x8214ba18.0xea50000.dmp	process.0x82067858.0xea50000.dmp	(100)
process.0x82159958.0xea50000.dmp	process.0x81ec2020.0xea50000.dmp	(100)
process.0x82159958.0xea50000.dmp	process.0x81f5e020.0xea50000.dmp	(100)
process.0x82159958.0xea50000.dmp	process.0x82067858.0xea50000.dmp	(100)
process.0x82159958.0xea50000.dmp	process.0x8214ba18.0xea50000.dmp	(100)
process.0x8226cda0.0xea50000.dmp	process.0x81ebd300.0xea50000.dmp	(100)
process.0x8226cda0.0xea50000.dmp	process.0x81f4b020.0xea50000.dmp	(100)
process.0x8226cda0.0xea50000.dmp	process.0x8207d5f0.0xea50000.dmp	(100)
process.0x82284b80.0xea50000.dmp	process.0x81ec2020.0xea50000.dmp	(100)

Filename 1	Filename 2	Match (in %)
process.0x82284b80.0xea50000.dmp	process.0x81f5e020.0xea50000.dmp	(100)
process.0x82284b80.0xea50000.dmp	process.0x82067858.0xea50000.dmp	(100)
process.0x82284b80.0xea50000.dmp	process.0x8214ba18.0xea50000.dmp	(100)
process.0x82284b80.0xea50000.dmp	process.0x82159958.0xea50000.dmp	(100)
process.0x822a0758.0xea50000.dmp	process.0x81ec2020.0xea50000.dmp	(100)
process.0x822a0758.0xea50000.dmp	process.0x81f5e020.0xea50000.dmp	(100)
process.0x822a0758.0xea50000.dmp	process.0x82067858.0xea50000.dmp	(100)
process.0x822a0758.0xea50000.dmp	process.0x8214ba18.0xea50000.dmp	(100)
process.0x822a0758.0xea50000.dmp	process.0x82159958.0xea50000.dmp	(100)
process.0x822a0758.0xea50000.dmp	process.0x82284b80.0xea50000.dmp	(100)
process.0x822e69f8.0xea50000.dmp	process.0x81ec2020.0xea50000.dmp	(100)
process.0x822e69f8.0xea50000.dmp	process.0x81f5e020.0xea50000.dmp	(100)
process.0x822e69f8.0xea50000.dmp	process.0x82067858.0xea50000.dmp	(100)
process.0x822e69f8.0xea50000.dmp	process.0x8214ba18.0xea50000.dmp	(100)
process.0x822e69f8.0xea50000.dmp	process.0x82159958.0xea50000.dmp	(100)
process.0x822e69f8.0xea50000.dmp	process.0x82284b80.0xea50000.dmp	(100)
process.0x822e69f8.0xea50000.dmp	process.0x822a0758.0xea50000.dmp	(100)
process.0x8230c5f8.0xea50000.dmp	process.0x81f7a708.0xea50000.dmp	(100)
process.0x82389020.0xea50000.dmp	process.0x81f7a708.0xea50000.dmp	(100)
process.0x82389020.0xea50000.dmp	process.0x8230c5f8.0xea50000.dmp	(100)
process.0x823f2020.0xea50000.dmp	process.0x81ebd300.0xea50000.dmp	(100)
process.0x823f2020.0xea50000.dmp	process.0x81f4b020.0xea50000.dmp	(100)
process.0x823f2020.0xea50000.dmp	process.0x8207d5f0.0xea50000.dmp	(100)
process.0x823f2020.0xea50000.dmp	process.0x8226cda0.0xea50000.dmp	(100)
process.0x82406da0.0xea50000.dmp	process.0x81ebd300.0xea50000.dmp	(100)
process.0x82406da0.0xea50000.dmp	process.0x81f4b020.0xea50000.dmp	(100)
process.0x82406da0.0xea50000.dmp	process.0x8207d5f0.0xea50000.dmp	(100)
process.0x82406da0.0xea50000.dmp	process.0x8226cda0.0xea50000.dmp	(100)
process.0x82406da0.0xea50000.dmp	process.0x823f2020.0xea50000.dmp	(100)
process.0x824264c0.0xea50000.dmp	process.0x81ebd300.0xea50000.dmp	(100)
process.0x824264c0.0xea50000.dmp	process.0x81f4b020.0xea50000.dmp	(100)

Filename 1	Filename 2	Match (in %)
process.0x824264c0.0xea50000.dmp	process.0x8207d5f0.0xea50000.dmp	(100)
process.0x824264c0.0xea50000.dmp	process.0x8226cda0.0xea50000.dmp	(100)
process.0x824264c0.0xea50000.dmp	process.0x823f2020.0xea50000.dmp	(100)
process.0x824264c0.0xea50000.dmp	process.0x82406da0.0xea50000.dmp	(100)
process.0x82436a48.0xea50000.dmp	process.0x81ebd300.0xea50000.dmp	(100)
process.0x82436a48.0xea50000.dmp	process.0x81f4b020.0xea50000.dmp	(100)
process.0x82436a48.0xea50000.dmp	process.0x8207d5f0.0xea50000.dmp	(100)
process.0x82436a48.0xea50000.dmp	process.0x8226cda0.0xea50000.dmp	(100)
process.0x82436a48.0xea50000.dmp	process.0x823f2020.0xea50000.dmp	(100)
process.0x82436a48.0xea50000.dmp	process.0x82406da0.0xea50000.dmp	(100)
process.0x82436a48.0xea50000.dmp	process.0x824264c0.0xea50000.dmp	(100)
process.0x824578b0.0xea50000.dmp	process.0x81f7a708.0xea50000.dmp	(100)
process.0x824578b0.0xea50000.dmp	process.0x8230c5f8.0xea50000.dmp	(100)
process.0x824578b0.0xea50000.dmp	process.0x82389020.0xea50000.dmp	(100)
process.0x82458020.0xea50000.dmp	process.0x81ebd300.0xea50000.dmp	(100)
process.0x82458020.0xea50000.dmp	process.0x81f4b020.0xea50000.dmp	(100)
process.0x82458020.0xea50000.dmp	process.0x8207d5f0.0xea50000.dmp	(100)
process.0x82458020.0xea50000.dmp	process.0x8226cda0.0xea50000.dmp	(100)
process.0x82458020.0xea50000.dmp	process.0x823f2020.0xea50000.dmp	(100)
process.0x82458020.0xea50000.dmp	process.0x82406da0.0xea50000.dmp	(100)
process.0x82458020.0xea50000.dmp	process.0x824264c0.0xea50000.dmp	(100)
process.0x82458020.0xea50000.dmp	process.0x82436a48.0xea50000.dmp	(100)
process.0x81ec2020.0xea50000.dmp	process.0x81ebd300.0xea50000.dmp	(99)
process.0x81f4b020.0xea50000.dmp	process.0x81ec2020.0xea50000.dmp	(99)
process.0x81f4c550.0xea50000.dmp	process.0x81ebd300.0xea50000.dmp	(99)
process.0x81f4c550.0xea50000.dmp	process.0x81ec2020.0xea50000.dmp	(99)
process.0x81f4c550.0xea50000.dmp	process.0x81f4b020.0xea50000.dmp	(99)
process.0x81f5e020.0xea50000.dmp	process.0x81ebd300.0xea50000.dmp	(99)
process.0x81f5e020.0xea50000.dmp	process.0x81f4b020.0xea50000.dmp	(99)
process.0x81f5e020.0xea50000.dmp	process.0x81f4c550.0xea50000.dmp	(99)
process.0x82067858.0xea50000.dmp	process.0x81ebd300.0xea50000.dmp	(99)

Filename 1	Filename 2	Match (in %)
process.0x82067858.0xea50000.dmp	process.0x81f4b020.0xea50000.dmp	(99)
process.0x82067858.0xea50000.dmp	process.0x81f4c550.0xea50000.dmp	(99)
process.0x82072660.0xea50000.dmp	process.0x81ebd300.0xea50000.dmp	(99)
process.0x82072660.0xea50000.dmp	process.0x81ec2020.0xea50000.dmp	(99)
process.0x82072660.0xea50000.dmp	process.0x81f4b020.0xea50000.dmp	(99)
process.0x82072660.0xea50000.dmp	process.0x81f4c550.0xea50000.dmp	(99)
process.0x82072660.0xea50000.dmp	process.0x81f5e020.0xea50000.dmp	(99)
process.0x82072660.0xea50000.dmp	process.0x82067858.0xea50000.dmp	(99)
process.0x8207d5f0.0xea50000.dmp	process.0x81ec2020.0xea50000.dmp	(99)
process.0x8207d5f0.0xea50000.dmp	process.0x81f4c550.0xea50000.dmp	(99)
process.0x8207d5f0.0xea50000.dmp	process.0x81f5e020.0xea50000.dmp	(99)
process.0x8207d5f0.0xea50000.dmp	process.0x82067858.0xea50000.dmp	(99)
process.0x8207d5f0.0xea50000.dmp	process.0x82072660.0xea50000.dmp	(99)
process.0x820bd760.0xeab0000.dmp	process.0x81f5e020.0xeab0000.dmp	(99)
process.0x8214ba18.0xea50000.dmp	process.0x81ebd300.0xea50000.dmp	(99)
process.0x8214ba18.0xea50000.dmp	process.0x81f4b020.0xea50000.dmp	(99)
process.0x8214ba18.0xea50000.dmp	process.0x81f4c550.0xea50000.dmp	(99)
process.0x8214ba18.0xea50000.dmp	process.0x82072660.0xea50000.dmp	(99)
process.0x8214ba18.0xea50000.dmp	process.0x8207d5f0.0xea50000.dmp	(99)
process.0x82159958.0xea50000.dmp	process.0x81ebd300.0xea50000.dmp	(99)
process.0x82159958.0xea50000.dmp	process.0x81f4b020.0xea50000.dmp	(99)
process.0x82159958.0xea50000.dmp	process.0x81f4c550.0xea50000.dmp	(99)
process.0x82159958.0xea50000.dmp	process.0x82072660.0xea50000.dmp	(99)
process.0x82159958.0xea50000.dmp	process.0x8207d5f0.0xea50000.dmp	(99)
process.0x8226cda0.0xea50000.dmp	process.0x81ec2020.0xea50000.dmp	(99)
process.0x8226cda0.0xea50000.dmp	process.0x81f4c550.0xea50000.dmp	(99)
process.0x8226cda0.0xea50000.dmp	process.0x81f5e020.0xea50000.dmp	(99)
process.0x8226cda0.0xea50000.dmp	process.0x82067858.0xea50000.dmp	(99)
process.0x8226cda0.0xea50000.dmp	process.0x82072660.0xea50000.dmp	(99)
process.0x8226cda0.0xea50000.dmp	process.0x8214ba18.0xea50000.dmp	(99)
process.0x8226cda0.0xea50000.dmp	process.0x82159958.0xea50000.dmp	(99)

Filename 1	Filename 2	Match (in %)
process.0x82284b80.0xea50000.dmp	process.0x81ebd300.0xea50000.dmp	(99)
process.0x82284b80.0xea50000.dmp	process.0x81f4b020.0xea50000.dmp	(99)
process.0x82284b80.0xea50000.dmp	process.0x81f4c550.0xea50000.dmp	(99)
process.0x82284b80.0xea50000.dmp	process.0x82072660.0xea50000.dmp	(99)
process.0x82284b80.0xea50000.dmp	process.0x8207d5f0.0xea50000.dmp	(99)
process.0x82284b80.0xea50000.dmp	process.0x8226cda0.0xea50000.dmp	(99)
process.0x822a0758.0xea50000.dmp	process.0x81ebd300.0xea50000.dmp	(99)
process.0x822a0758.0xea50000.dmp	process.0x81f4b020.0xea50000.dmp	(99)
process.0x822a0758.0xea50000.dmp	process.0x81f4c550.0xea50000.dmp	(99)
process.0x822a0758.0xea50000.dmp	process.0x82072660.0xea50000.dmp	(99)
process.0x822a0758.0xea50000.dmp	process.0x8207d5f0.0xea50000.dmp	(99)
process.0x822a0758.0xea50000.dmp	process.0x8226cda0.0xea50000.dmp	(99)
process.0x822e69f8.0xea50000.dmp	process.0x81ebd300.0xea50000.dmp	(99)
process.0x822e69f8.0xea50000.dmp	process.0x81f4b020.0xea50000.dmp	(99)
process.0x822e69f8.0xea50000.dmp	process.0x81f4c550.0xea50000.dmp	(99)
process.0x822e69f8.0xea50000.dmp	process.0x82072660.0xea50000.dmp	(99)
process.0x822e69f8.0xea50000.dmp	process.0x8207d5f0.0xea50000.dmp	(99)
process.0x822e69f8.0xea50000.dmp	process.0x8226cda0.0xea50000.dmp	(99)
process.0x8236d7a0.0xea50000.dmp	process.0x81ec2020.0xea50000.dmp	(99)
process.0x8236d7a0.0xea50000.dmp	process.0x81f5e020.0xea50000.dmp	(99)
process.0x8236d7a0.0xea50000.dmp	process.0x81f7a708.0xea50000.dmp	(99)
process.0x8236d7a0.0xea50000.dmp	process.0x82067858.0xea50000.dmp	(99)
process.0x8236d7a0.0xea50000.dmp	process.0x8214ba18.0xea50000.dmp	(99)
process.0x8236d7a0.0xea50000.dmp	process.0x82159958.0xea50000.dmp	(99)
process.0x8236d7a0.0xea50000.dmp	process.0x82284b80.0xea50000.dmp	(99)
process.0x8236d7a0.0xea50000.dmp	process.0x822a0758.0xea50000.dmp	(99)
process.0x8236d7a0.0xea50000.dmp	process.0x822e69f8.0xea50000.dmp	(99)
process.0x8236d7a0.0xea50000.dmp	process.0x8230c5f8.0xea50000.dmp	(99)
process.0x82389020.0xea50000.dmp	process.0x8236d7a0.0xea50000.dmp	(99)
process.0x823f2020.0xea50000.dmp	process.0x81ec2020.0xea50000.dmp	(99)
process.0x823f2020.0xea50000.dmp	process.0x81f4c550.0xea50000.dmp	(99)

Filename 1	Filename 2	Match (in %)
process.0x823f2020.0xea50000.dmp	process.0x81f5e020.0xea50000.dmp	(99)
process.0x823f2020.0xea50000.dmp	process.0x82067858.0xea50000.dmp	(99)
process.0x823f2020.0xea50000.dmp	process.0x82072660.0xea50000.dmp	(99)
process.0x823f2020.0xea50000.dmp	process.0x8214ba18.0xea50000.dmp	(99)
process.0x823f2020.0xea50000.dmp	process.0x82159958.0xea50000.dmp	(99)
process.0x823f2020.0xea50000.dmp	process.0x82284b80.0xea50000.dmp	(99)
process.0x823f2020.0xea50000.dmp	process.0x822a0758.0xea50000.dmp	(99)
process.0x823f2020.0xea50000.dmp	process.0x822e69f8.0xea50000.dmp	(99)
process.0x82406da0.0xea50000.dmp	process.0x81ec2020.0xea50000.dmp	(99)
process.0x82406da0.0xea50000.dmp	process.0x81f4c550.0xea50000.dmp	(99)
process.0x82406da0.0xea50000.dmp	process.0x81f5e020.0xea50000.dmp	(99)
process.0x82406da0.0xea50000.dmp	process.0x82067858.0xea50000.dmp	(99)
process.0x82406da0.0xea50000.dmp	process.0x82072660.0xea50000.dmp	(99)
process.0x82406da0.0xea50000.dmp	process.0x8214ba18.0xea50000.dmp	(99)
process.0x82406da0.0xea50000.dmp	process.0x82159958.0xea50000.dmp	(99)
process.0x82406da0.0xea50000.dmp	process.0x82284b80.0xea50000.dmp	(99)
process.0x82406da0.0xea50000.dmp	process.0x822a0758.0xea50000.dmp	(99)
process.0x82406da0.0xea50000.dmp	process.0x822e69f8.0xea50000.dmp	(99)
process.0x824264c0.0xea50000.dmp	process.0x81ec2020.0xea50000.dmp	(99)
process.0x824264c0.0xea50000.dmp	process.0x81f4c550.0xea50000.dmp	(99)
process.0x824264c0.0xea50000.dmp	process.0x81f5e020.0xea50000.dmp	(99)
process.0x824264c0.0xea50000.dmp	process.0x82067858.0xea50000.dmp	(99)
process.0x824264c0.0xea50000.dmp	process.0x82072660.0xea50000.dmp	(99)
process.0x824264c0.0xea50000.dmp	process.0x8214ba18.0xea50000.dmp	(99)
process.0x824264c0.0xea50000.dmp	process.0x82159958.0xea50000.dmp	(99)
process.0x824264c0.0xea50000.dmp	process.0x82284b80.0xea50000.dmp	(99)
process.0x824264c0.0xea50000.dmp	process.0x822a0758.0xea50000.dmp	(99)
process.0x824264c0.0xea50000.dmp	process.0x822e69f8.0xea50000.dmp	(99)
process.0x82436a48.0xea50000.dmp	process.0x81ec2020.0xea50000.dmp	(99)
process.0x82436a48.0xea50000.dmp	process.0x81f4c550.0xea50000.dmp	(99)
process.0x82436a48.0xea50000.dmp	process.0x81f5e020.0xea50000.dmp	(99)

Filename 1	Filename 2	Match (in %)
process.0x82436a48.0xea50000.dmp	process.0x82067858.0xea50000.dmp	(99)
process.0x82436a48.0xea50000.dmp	process.0x82072660.0xea50000.dmp	(99)
process.0x82436a48.0xea50000.dmp	process.0x8214ba18.0xea50000.dmp	(99)
process.0x82436a48.0xea50000.dmp	process.0x82159958.0xea50000.dmp	(99)
process.0x82436a48.0xea50000.dmp	process.0x82284b80.0xea50000.dmp	(99)
process.0x82436a48.0xea50000.dmp	process.0x822a0758.0xea50000.dmp	(99)
process.0x82436a48.0xea50000.dmp	process.0x822e69f8.0xea50000.dmp	(99)
process.0x824578b0.0xea50000.dmp	process.0x8236d7a0.0xea50000.dmp	(99)
process.0x82458020.0xea50000.dmp	process.0x81ec2020.0xea50000.dmp	(99)
process.0x82458020.0xea50000.dmp	process.0x81f4c550.0xea50000.dmp	(99)
process.0x82458020.0xea50000.dmp	process.0x81f5e020.0xea50000.dmp	(99)
process.0x82458020.0xea50000.dmp	process.0x82067858.0xea50000.dmp	(99)
process.0x82458020.0xea50000.dmp	process.0x82072660.0xea50000.dmp	(99)
process.0x82458020.0xea50000.dmp	process.0x8214ba18.0xea50000.dmp	(99)
process.0x82458020.0xea50000.dmp	process.0x82159958.0xea50000.dmp	(99)
process.0x82458020.0xea50000.dmp	process.0x82284b80.0xea50000.dmp	(99)
process.0x82458020.0xea50000.dmp	process.0x822a0758.0xea50000.dmp	(99)
process.0x82458020.0xea50000.dmp	process.0x822e69f8.0xea50000.dmp	(99)
process.0x81f7a708.0xea50000.dmp	process.0x81ebd300.0xea50000.dmp	(97)
process.0x81f7a708.0xea50000.dmp	process.0x81ec2020.0xea50000.dmp	(97)
process.0x81f7a708.0xea50000.dmp	process.0x81f4b020.0xea50000.dmp	(97)
process.0x81f7a708.0xea50000.dmp	process.0x81f4c550.0xea50000.dmp	(97)
process.0x81f7a708.0xea50000.dmp	process.0x81f5e020.0xea50000.dmp	(97)
process.0x82067858.0xea50000.dmp	process.0x81f7a708.0xea50000.dmp	(97)
process.0x82072660.0xea50000.dmp	process.0x81f7a708.0xea50000.dmp	(97)
process.0x8207d5f0.0xea50000.dmp	process.0x81f7a708.0xea50000.dmp	(97)
process.0x8214ba18.0xea50000.dmp	process.0x81f7a708.0xea50000.dmp	(97)
process.0x82159958.0xea50000.dmp	process.0x81f7a708.0xea50000.dmp	(97)
process.0x8226cda0.0xea50000.dmp	process.0x81f7a708.0xea50000.dmp	(97)
process.0x82284b80.0xea50000.dmp	process.0x81f7a708.0xea50000.dmp	(97)
process.0x822a0758.0xea50000.dmp	process.0x81f7a708.0xea50000.dmp	(97)

Filename 1	Filename 2	Match (in %)
process.0x822e69f8.0xea50000.dmp	process.0x81f7a708.0xea50000.dmp	(97)
process.0x8230c5f8.0xea50000.dmp	process.0x81ebd300.0xea50000.dmp	(97)
process.0x8230c5f8.0xea50000.dmp	process.0x81ec2020.0xea50000.dmp	(97)
process.0x8230c5f8.0xea50000.dmp	process.0x81f4b020.0xea50000.dmp	(97)
process.0x8230c5f8.0xea50000.dmp	process.0x81f4c550.0xea50000.dmp	(97)
process.0x8230c5f8.0xea50000.dmp	process.0x81f5e020.0xea50000.dmp	(97)
process.0x8230c5f8.0xea50000.dmp	process.0x82067858.0xea50000.dmp	(97)
process.0x8230c5f8.0xea50000.dmp	process.0x82072660.0xea50000.dmp	(97)
process.0x8230c5f8.0xea50000.dmp	process.0x8207d5f0.0xea50000.dmp	(97)
process.0x8230c5f8.0xea50000.dmp	process.0x8214ba18.0xea50000.dmp	(97)
process.0x8230c5f8.0xea50000.dmp	process.0x82159958.0xea50000.dmp	(97)
process.0x8230c5f8.0xea50000.dmp	process.0x8226cda0.0xea50000.dmp	(97)
process.0x8230c5f8.0xea50000.dmp	process.0x82284b80.0xea50000.dmp	(97)
process.0x8230c5f8.0xea50000.dmp	process.0x822a0758.0xea50000.dmp	(97)
process.0x8230c5f8.0xea50000.dmp	process.0x822e69f8.0xea50000.dmp	(97)
process.0x8236d7a0.0xea50000.dmp	process.0x81ebd300.0xea50000.dmp	(97)
process.0x8236d7a0.0xea50000.dmp	process.0x81f4b020.0xea50000.dmp	(97)
process.0x8236d7a0.0xea50000.dmp	process.0x81f4c550.0xea50000.dmp	(97)
process.0x8236d7a0.0xea50000.dmp	process.0x82072660.0xea50000.dmp	(97)
process.0x8236d7a0.0xea50000.dmp	process.0x8207d5f0.0xea50000.dmp	(97)
process.0x8236d7a0.0xea50000.dmp	process.0x8226cda0.0xea50000.dmp	(97)
process.0x82389020.0xea50000.dmp	process.0x81ebd300.0xea50000.dmp	(97)
process.0x82389020.0xea50000.dmp	process.0x81ec2020.0xea50000.dmp	(97)
process.0x82389020.0xea50000.dmp	process.0x81f4b020.0xea50000.dmp	(97)
process.0x82389020.0xea50000.dmp	process.0x81f4c550.0xea50000.dmp	(97)
process.0x82389020.0xea50000.dmp	process.0x81f5e020.0xea50000.dmp	(97)
process.0x82389020.0xea50000.dmp	process.0x82067858.0xea50000.dmp	(97)
process.0x82389020.0xea50000.dmp	process.0x82072660.0xea50000.dmp	(97)
process.0x82389020.0xea50000.dmp	process.0x8207d5f0.0xea50000.dmp	(97)
process.0x82389020.0xea50000.dmp	process.0x8214ba18.0xea50000.dmp	(97)
process.0x82389020.0xea50000.dmp	process.0x82159958.0xea50000.dmp	(97)

Filename 1	Filename 2	Match (in %)
process.0x82389020.0xea50000.dmp	process.0x8226cda0.0xea50000.dmp	(97)
process.0x82389020.0xea50000.dmp	process.0x82284b80.0xea50000.dmp	(97)
process.0x82389020.0xea50000.dmp	process.0x822a0758.0xea50000.dmp	(97)
process.0x82389020.0xea50000.dmp	process.0x822e69f8.0xea50000.dmp	(97)
process.0x823f2020.0xea50000.dmp	process.0x81f7a708.0xea50000.dmp	(97)
process.0x823f2020.0xea50000.dmp	process.0x8230c5f8.0xea50000.dmp	(97)
process.0x823f2020.0xea50000.dmp	process.0x8236d7a0.0xea50000.dmp	(97)
process.0x823f2020.0xea50000.dmp	process.0x82389020.0xea50000.dmp	(97)
process.0x82406da0.0xea50000.dmp	process.0x81f7a708.0xea50000.dmp	(97)
process.0x82406da0.0xea50000.dmp	process.0x8230c5f8.0xea50000.dmp	(97)
process.0x82406da0.0xea50000.dmp	process.0x8236d7a0.0xea50000.dmp	(97)
process.0x82406da0.0xea50000.dmp	process.0x82389020.0xea50000.dmp	(97)
process.0x824264c0.0xea50000.dmp	process.0x81f7a708.0xea50000.dmp	(97)
process.0x824264c0.0xea50000.dmp	process.0x8230c5f8.0xea50000.dmp	(97)
process.0x824264c0.0xea50000.dmp	process.0x8236d7a0.0xea50000.dmp	(97)
process.0x824264c0.0xea50000.dmp	process.0x82389020.0xea50000.dmp	(97)
process.0x82436a48.0xea50000.dmp	process.0x81f7a708.0xea50000.dmp	(97)
process.0x82436a48.0xea50000.dmp	process.0x8230c5f8.0xea50000.dmp	(97)
process.0x82436a48.0xea50000.dmp	process.0x8236d7a0.0xea50000.dmp	(97)
process.0x82436a48.0xea50000.dmp	process.0x82389020.0xea50000.dmp	(97)
process.0x824578b0.0xea50000.dmp	process.0x81ebd300.0xea50000.dmp	(97)
process.0x824578b0.0xea50000.dmp	process.0x81ec2020.0xea50000.dmp	(97)
process.0x824578b0.0xea50000.dmp	process.0x81f4b020.0xea50000.dmp	(97)
process.0x824578b0.0xea50000.dmp	process.0x81f4c550.0xea50000.dmp	(97)
process.0x824578b0.0xea50000.dmp	process.0x81f5e020.0xea50000.dmp	(97)
process.0x824578b0.0xea50000.dmp	process.0x82067858.0xea50000.dmp	(97)
process.0x824578b0.0xea50000.dmp	process.0x82072660.0xea50000.dmp	(97)
process.0x824578b0.0xea50000.dmp	process.0x8207d5f0.0xea50000.dmp	(97)
process.0x824578b0.0xea50000.dmp	process.0x8214ba18.0xea50000.dmp	(97)
process.0x824578b0.0xea50000.dmp	process.0x82159958.0xea50000.dmp	(97)
process.0x824578b0.0xea50000.dmp	process.0x8226cda0.0xea50000.dmp	(97)

Filename 1	Filename 2	Match (in %)
process.0x824578b0.0xea50000.dmp	process.0x82284b80.0xea50000.dmp	(97)
process.0x824578b0.0xea50000.dmp	process.0x822a0758.0xea50000.dmp	(97)
process.0x824578b0.0xea50000.dmp	process.0x822e69f8.0xea50000.dmp	(97)
process.0x824578b0.0xea50000.dmp	process.0x823f2020.0xea50000.dmp	(97)
process.0x824578b0.0xea50000.dmp	process.0x82406da0.0xea50000.dmp	(97)
process.0x824578b0.0xea50000.dmp	process.0x824264c0.0xea50000.dmp	(97)
process.0x824578b0.0xea50000.dmp	process.0x82436a48.0xea50000.dmp	(97)
process.0x82458020.0xea50000.dmp	process.0x81f7a708.0xea50000.dmp	(97)
process.0x82458020.0xea50000.dmp	process.0x8230c5f8.0xea50000.dmp	(97)
process.0x82458020.0xea50000.dmp	process.0x8236d7a0.0xea50000.dmp	(97)
process.0x82458020.0xea50000.dmp	process.0x82389020.0xea50000.dmp	(97)
process.0x82458020.0xea50000.dmp	process.0x824578b0.0xea50000.dmp	(97)
process.0x823e32f8.0xea50000.dmp	process.0x81f4c550.0xea50000.dmp	(96)
process.0x823e32f8.0xea50000.dmp	process.0x81f7a708.0xea50000.dmp	(96)
process.0x823e32f8.0xea50000.dmp	process.0x82072660.0xea50000.dmp	(96)
process.0x823e32f8.0xea50000.dmp	process.0x8230c5f8.0xea50000.dmp	(96)
process.0x823e32f8.0xea50000.dmp	process.0x8236d7a0.0xea50000.dmp	(96)
process.0x823e32f8.0xea50000.dmp	process.0x82389020.0xea50000.dmp	(96)
process.0x823e32f8.0xeab0000.dmp	process.0x820bd760.0xeab0000.dmp	(96)
process.0x824578b0.0xea50000.dmp	process.0x823e32f8.0xea50000.dmp	(96)
process.0x823e32f8.0xea50000.dmp	process.0x81ebd300.0xea50000.dmp	(94)
process.0x823e32f8.0xea50000.dmp	process.0x81ec2020.0xea50000.dmp	(94)
process.0x823e32f8.0xea50000.dmp	process.0x81f4b020.0xea50000.dmp	(94)
process.0x823e32f8.0xea50000.dmp	process.0x81f5e020.0xea50000.dmp	(94)
process.0x823e32f8.0xea50000.dmp	process.0x82067858.0xea50000.dmp	(94)
process.0x823e32f8.0xea50000.dmp	process.0x8207d5f0.0xea50000.dmp	(94)
process.0x823e32f8.0xea50000.dmp	process.0x8214ba18.0xea50000.dmp	(94)
process.0x823e32f8.0xea50000.dmp	process.0x82159958.0xea50000.dmp	(94)
process.0x823e32f8.0xea50000.dmp	process.0x8226cda0.0xea50000.dmp	(94)
process.0x823e32f8.0xea50000.dmp	process.0x82284b80.0xea50000.dmp	(94)
process.0x823e32f8.0xea50000.dmp	process.0x822a0758.0xea50000.dmp	(94)

Filename 1	Filename 2	Match (in %)
process.0x823e32f8.0xea50000.dmp	process.0x822e69f8.0xea50000.dmp	(94)
process.0x823e32f8.0xeab0000.dmp	process.0x81f5e020.0xeab0000.dmp	(94)
process.0x823f2020.0xea50000.dmp	process.0x823e32f8.0xea50000.dmp	(94)
process.0x82406da0.0xea50000.dmp	process.0x823e32f8.0xea50000.dmp	(94)
process.0x824264c0.0xea50000.dmp	process.0x823e32f8.0xea50000.dmp	(94)
process.0x82436a48.0xea50000.dmp	process.0x823e32f8.0xea50000.dmp	(94)
process.0x82458020.0xea50000.dmp	process.0x823e32f8.0xea50000.dmp	(94)
process.0x823e32f8.0xeab0000.dmp	process.0x823e32f8.0xea50000.dmp	(40)
process.0x81f5e020.0xeab0000.dmp	process.0x81ebd300.0xea50000.dmp	(38)
process.0x81f5e020.0xeab0000.dmp	process.0x81ec2020.0xea50000.dmp	(38)
process.0x81f5e020.0xeab0000.dmp	process.0x81f4b020.0xea50000.dmp	(38)
process.0x81f5e020.0xeab0000.dmp	process.0x81f4c550.0xea50000.dmp	(38)
process.0x81f5e020.0xeab0000.dmp	process.0x81f5e020.0xea50000.dmp	(38)
process.0x81f7a708.0xea50000.dmp	process.0x81f5e020.0xeab0000.dmp	(38)
process.0x82067858.0xea50000.dmp	process.0x81f5e020.0xeab0000.dmp	(38)
process.0x82072660.0xea50000.dmp	process.0x81f5e020.0xeab0000.dmp	(38)
process.0x8207d5f0.0xea50000.dmp	process.0x81f5e020.0xeab0000.dmp	(38)
process.0x820bd760.0xeab0000.dmp	process.0x81ebd300.0xea50000.dmp	(38)
process.0x820bd760.0xeab0000.dmp	process.0x81ec2020.0xea50000.dmp	(38)
process.0x820bd760.0xeab0000.dmp	process.0x81f4b020.0xea50000.dmp	(38)
process.0x820bd760.0xeab0000.dmp	process.0x81f4c550.0xea50000.dmp	(38)
process.0x820bd760.0xeab0000.dmp	process.0x81f5e020.0xea50000.dmp	(38)
process.0x820bd760.0xeab0000.dmp	process.0x81f7a708.0xea50000.dmp	(38)
process.0x820bd760.0xeab0000.dmp	process.0x82067858.0xea50000.dmp	(38)
process.0x820bd760.0xeab0000.dmp	process.0x82072660.0xea50000.dmp	(38)
process.0x820bd760.0xeab0000.dmp	process.0x8207d5f0.0xea50000.dmp	(38)
process.0x8214ba18.0xea50000.dmp	process.0x81f5e020.0xeab0000.dmp	(38)
process.0x8214ba18.0xea50000.dmp	process.0x820bd760.0xeab0000.dmp	(38)
process.0x82159958.0xea50000.dmp	process.0x81f5e020.0xeab0000.dmp	(38)
process.0x82159958.0xea50000.dmp	process.0x820bd760.0xeab0000.dmp	(38)
process.0x8226cda0.0xea50000.dmp	process.0x81f5e020.0xeab0000.dmp	(38)

Filename 1	Filename 2	Match (in %)
process.0x8226cda0.0xea50000.dmp	process.0x820bd760.0xeab0000.dmp	(38)
process.0x82284b80.0xea50000.dmp	process.0x81f5e020.0xeab0000.dmp	(38)
process.0x82284b80.0xea50000.dmp	process.0x820bd760.0xeab0000.dmp	(38)
process.0x822a0758.0xea50000.dmp	process.0x81f5e020.0xeab0000.dmp	(38)
process.0x822a0758.0xea50000.dmp	process.0x820bd760.0xeab0000.dmp	(38)
process.0x822e69f8.0xea50000.dmp	process.0x81f5e020.0xeab0000.dmp	(38)
process.0x822e69f8.0xea50000.dmp	process.0x820bd760.0xeab0000.dmp	(38)
process.0x8230c5f8.0xea50000.dmp	process.0x81f5e020.0xeab0000.dmp	(38)
process.0x8230c5f8.0xea50000.dmp	process.0x820bd760.0xeab0000.dmp	(38)
process.0x8236d7a0.0xea50000.dmp	process.0x81f5e020.0xeab0000.dmp	(38)
process.0x8236d7a0.0xea50000.dmp	process.0x820bd760.0xeab0000.dmp	(38)
process.0x82389020.0xea50000.dmp	process.0x81f5e020.0xeab0000.dmp	(38)
process.0x82389020.0xea50000.dmp	process.0x820bd760.0xeab0000.dmp	(38)
process.0x823f2020.0xea50000.dmp	process.0x81f5e020.0xeab0000.dmp	(38)
process.0x823f2020.0xea50000.dmp	process.0x820bd760.0xeab0000.dmp	(38)
process.0x82406da0.0xea50000.dmp	process.0x81f5e020.0xeab0000.dmp	(38)
process.0x82406da0.0xea50000.dmp	process.0x820bd760.0xeab0000.dmp	(38)
process.0x824264c0.0xea50000.dmp	process.0x81f5e020.0xeab0000.dmp	(38)
process.0x824264c0.0xea50000.dmp	process.0x820bd760.0xeab0000.dmp	(38)
process.0x82436a48.0xea50000.dmp	process.0x81f5e020.0xeab0000.dmp	(38)
process.0x82436a48.0xea50000.dmp	process.0x820bd760.0xeab0000.dmp	(38)
process.0x824578b0.0xea50000.dmp	process.0x81f5e020.0xeab0000.dmp	(38)
process.0x824578b0.0xea50000.dmp	process.0x820bd760.0xeab0000.dmp	(38)
process.0x82458020.0xea50000.dmp	process.0x81f5e020.0xeab0000.dmp	(38)
process.0x82458020.0xea50000.dmp	process.0x820bd760.0xeab0000.dmp	(38)
process.0x823e32f8.0xea50000.dmp	process.0x81f5e020.0xeab0000.dmp	(36)
process.0x823e32f8.0xea50000.dmp	process.0x820bd760.0xeab0000.dmp	(36)
process.0x823e32f8.0xeab0000.dmp	process.0x81ebd300.0xea50000.dmp	(36)
process.0x823e32f8.0xeab0000.dmp	process.0x81ec2020.0xea50000.dmp	(36)
process.0x823e32f8.0xeab0000.dmp	process.0x81f4b020.0xea50000.dmp	(36)
process.0x823e32f8.0xeab0000.dmp	process.0x81f4c550.0xea50000.dmp	(36)

Filename 1	Filename 2	Match (in %)
process.0x823e32f8.0xeab0000.dmp	process.0x81f5e020.0xea50000.dmp	(36)
process.0x823e32f8.0xeab0000.dmp	process.0x81f7a708.0xea50000.dmp	(36)
process.0x823e32f8.0xeab0000.dmp	process.0x82067858.0xea50000.dmp	(36)
process.0x823e32f8.0xeab0000.dmp	process.0x82072660.0xea50000.dmp	(36)
process.0x823e32f8.0xeab0000.dmp	process.0x8207d5f0.0xea50000.dmp	(36)
process.0x823e32f8.0xeab0000.dmp	process.0x8214ba18.0xea50000.dmp	(36)
process.0x823e32f8.0xeab0000.dmp	process.0x82159958.0xea50000.dmp	(36)
process.0x823e32f8.0xeab0000.dmp	process.0x8226cda0.0xea50000.dmp	(36)
process.0x823e32f8.0xeab0000.dmp	process.0x82284b80.0xea50000.dmp	(36)
process.0x823e32f8.0xeab0000.dmp	process.0x822a0758.0xea50000.dmp	(36)
process.0x823e32f8.0xeab0000.dmp	process.0x822e69f8.0xea50000.dmp	(36)
process.0x823e32f8.0xeab0000.dmp	process.0x8230c5f8.0xea50000.dmp	(36)
process.0x823e32f8.0xeab0000.dmp	process.0x8236d7a0.0xea50000.dmp	(36)
process.0x823e32f8.0xeab0000.dmp	process.0x82389020.0xea50000.dmp	(36)
process.0x823f2020.0xea50000.dmp	process.0x823e32f8.0xeab0000.dmp	(36)
process.0x82406da0.0xea50000.dmp	process.0x823e32f8.0xeab0000.dmp	(36)
process.0x824264c0.0xea50000.dmp	process.0x823e32f8.0xeab0000.dmp	(36)
process.0x82436a48.0xea50000.dmp	process.0x823e32f8.0xeab0000.dmp	(36)
process.0x824578b0.0xea50000.dmp	process.0x823e32f8.0xeab0000.dmp	(36)
process.0x82458020.0xea50000.dmp	process.0x823e32f8.0xeab0000.dmp	(36)

This page intentionally left blank.

Bibliography

Carbone, Richard. Malware memory analysis for non-specialists: Investigating a publicly available memory image of the Zeus Trojan horse. Technical Memorandum. Defence R&D Canada – Valcartier. TM 2013-018. April 2013.

Volatility. CommandReference: Example usage cases and output for Volatility 2.0 commands. Online command reference. Volatility. February 2012. http://code.google.com/p/volatility/wiki/CommandReference.

List of symbols/abbreviations/acronyms/initialisms

ASCII	American Standard Code for Information Interchange
AV	Anti-Virus or Antivirus
CFNOC	Canadian Forces Network Operations Centre
CORFC	Centre d'opérations des réseaux des Forces canadiennes
CTPH	Context Triggered Piecewise Hash Sometimes known as fuzzy hash or *ssdeep* hash
DLL	Dynamically Loaded Library
DND	Department of National Defence
DRDC	Defence Research & Development Canada
DRDKIM	Director Research and Development Knowledge and Information Management
EDT	Eastern Daylight Time
EXT4	Fourth Extended Filesystem
FTP	File Transfer Protocol
GICT	Groupe intégré de la criminalité technologique
GRC	Gendarmerie Royale du Canada
GRE	Generic Routing Encapsulation
HKCU	HKEY_LOCAL_USER
HKLM	HKEY_LOCAL_MACHINE
ID	Identification
IP	Internet Protocol
ITCU	Integrated Technological Crime Unit
MAC	Mandatory Access Control
MiB	Mebibyte
N/A	Not Available
NIST	National Institute of Standards and Technology
NSRL	National Software Reference Library
NTP	Network Time Protocol
PAE	Physical Address Extension
PE	Portable Executable

PID	Process ID
PPID	Parent Process ID
R&D	Research & Development
RAM	Random Access Memory
RCMP	Royal Canadian Mounted Police
RDDC	Recherche et Développement pour la Défense Canada
RTSP	Real Time Streaming Protocol
SHA1	Secure Hash Algorithm 1
SIP	Session Initiation Protocol
TCP	Transmission Control Protocol
TID	Thread ID
UDP	User Datagram Protocol
UPX	Ultimate Packer for eXecutables
URL	Uniform Resource Locator
UTC	Coordinated Universal Time
VAD	Virtual Address Descriptor
VMEM	Virtual Memory
WPF	Windows Presentation Foundation

Glossary

_Eprocess

See Eprocess.

Anti-Virus

An Anti-virus, AV, or AV scanner is a software system or framework which is used to, at a minimum, scan a given system for signs of malware infection. This software may be more than just a scanner; it may also include system-protection and anti-malware detection and prevention capability.

AV Scanner

See Anti-Virus.

Computer Memory Image

See Memory Image.

Context Triggered Piecewise Hash

See Fuzzy Hash.

Data Carving

Commonly known as file carving, data carving is the process or act of recovering known data structures, generally based on recognized file patterns. Data carving only works on contiguous data structures as the recovery of fragmented data is not supported by most of today's data recovery software and those that do support a very limited number of file formats.

Eprocess

The Eprocess is a kernel-based process-specific data structure that encompasses a process' state-based information. This structure has a forward and backward pointer to active processes.

Ext4

Ext4 is the latest Ext-based filesystem of the Linux operating system and supersedes Ext2/3. It provides filesystem journaling and greater performance, reliability and allows for much larger file and filesystem sizes. This filesystem is natively supported by Linux.

Fuzzy Hash

This is a specific type of file hashing which has the ability to identify file similarities, usually represented as a percentage. It is based on Context Triggered Piecewise Hashing, first proposed by Dr. Andrew Tridgell.

Handle

A handle is a pointer-like resource-based reference used to a specific system resource. Handles are abstract references to resources available within a given computer system. Under Windows, many types of handles exist but common examples pertain to files, directories, the registry and system based devices. It should not be confused with file handles.

Hash

A hash, commonly referred to as a file hash, is a reduced representation of some arbitrary data produced by passing it through some cryptographic hashing algorithm. In so doing, a unique hash value is generated by the hashing program and it can be used to identify and authenticate a given file's integrity and uniqueness against a set of hashes, commonly known as a hash-set. SHA1 and CTPH hashes are examples of hashing algorithms.

Memory Image

A memory image or computer memory image is a bit-copy of a computer system's RAM and is acquired using a memory-imaging program. In virtualized environments, memory can be acquired by an imaging program or by saving or dumping the virtual machine's memory state.

Mutex

A mutex is a Windows-based object used to provide exclusive access to a shared system resource. These resources can only be accessed one at a time, thus by issuing a mutex or mutual exclusion, a process or thread can be allocated said resource when it becomes available for use.

SHA1

The SHA1 hash is a 160-bit cryptographic hash commonly used for forensic file identification and authentication.

Strings Command

The *strings* command is a UNIX-based command used to extract 7, 8, 16 and 32-bit text patterns from arbitrary data files that are text or binary based. 7-bit extraction represents the first 128 ASCII characters while 8-bit extraction represents the extended ASCII character set. 16 and 32-bit strings are typically reserved for Unicode-based text. Thus, the command line parameters required to instruct the *strings* command to perform 7, 8, 16 or 32-bit text extraction are *-s*, *-S*, *-l* and *-L*, respectively.

Thread

A thread is typically a subset process. A thread contains only the code necessary to perform a set of instructions. In single-threaded programs, a thread represents the program's executable code and stack while in multi-threaded applications a thread performs just one piece of the work that is distributed across multiple threads. These threads then typically communicate with each other through various inter-process mechanisms.

Trojan horse

A Trojan horse is a malicious non-replicating infectious computer program. It infects a computer when the delivery software is run at which time a payload is instantiated that does the actual infecting. However, Trojan's do not typically infect computers the way viruses do. As such, they do not generally infect computer files. The program delivering the payload is known as a dropper. The payload achieves its objective by gaining some form of administrative level privileges in the target's operating system, typically through subversion. A Trojan's typical objective is to provide backdoor access but it can also be used for other capabilities including data and information theft, arbitrary or specific data file encryption and it can inflict damage to the operating system or its data files. In rare cases, it can even attempt to damage a system's hardware components.

UPX

UPX is an open source data compression algorithm used to compress executable files. UPX executable file packers exist for Windows, Linux, Mac OS X and other platforms.

Vmem

A Vmem file is a VMware virtual machine-based paged memory file. It is generated when a virtual machine's state is saved and contains the entire RAM allocated to that virtual machine.

Worm

Sometimes known as a computer or network worm, a worm is a malicious program designed to spread to as many computer systems as possible, usually by means of a network. Worms do not typically cause much, if any, damage to the underlying computer system. Instead, due to their need to replicate they often consume not only a network's available bandwidth but crash underlying computer systems as they sometimes overwhelm a system's resources as it attempts to propagate. Worms typically spread only to systems susceptible to the vulnerabilities necessary for their infection to take hold. Thus, unaffected systems do not become infected.

DOCUMENT CONTROL DATA

(Security markings for the title, abstract and indexing annotation must be entered when the document is Classified or Designated)

1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)

Defence Research and Development Canada – Valcartier
2459 Pie-XI Blvd North
Quebec (Quebec)
G3J 1X5 Canada

2a. SECURITY MARKING (Overall security marking of the document including special supplemental markings if applicable.)

UNCLASSIFIED

2b. CONTROLLED GOODS

(NON-CONTROLLED GOODS)
DMC A
REVIEW: GCEC APRIL 2011

3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title.)

Malware memory analysis for non-specialists : Investigating publicly available memory images for Prolaco and SpyEye

4. AUTHORS (last name, followed by initials – ranks, titles, etc. not to be used)

Carbone, R.

5. DATE OF PUBLICATION (Month and year of publication of document.)	6a. NO. OF PAGES (Total containing information, including Annexes, Appendices, etc.)	6b. NO. OF REFS (Total cited in document.)
October 2013	120	23

7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Technical Memorandum

8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.)

Defence Research and Development Canada – Valcartier
2459 Pie-XI Blvd North
Quebec (Quebec)
G3J 1X5 Canada

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)	9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.)
31XF20 « MOU RCMP "Live Forensics" »	

10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.)	10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)
DRDC Valcartier TM 2013-155	

11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.)

Unlimited

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11) is possible, a wider announcement audience may be selected.))

Unlimited

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

This technical memorandum examines how an investigator can analyse an infected Windows memory dump. The author investigates how to carry out such an analysis using Volatility and other investigative tools, including data carving utilities and anti-virus scanners. Volatility is a popular and evolving open source-based memory analysis framework upon which the author has proposed a memory-specific methodology for aiding fellow novice memory analysts. The author examines how Volatility can be used to find evidence and indicators of infection. This technical memorandum is the second in a series concerning Windows malware-based memory analysis. This current work examines two memory images infected with Prolaco and SpyEye, respectively.

Ce mémorandum technique examine comment un investigateur peut analyser une image mémoire d'une machine Windows infectée. L'auteur investigue les techniques d'analyse utilisant Volatility et d'autres outils tels que les utilitaires de récupération de données et les scanneurs anti-virus. Volatility est un cadre populaire d'analyse de mémoire en source libre sur lequel l'auteur s'appuie pour proposer une méthodologie spécifique à la mémoire pour aider ses collègues analystes novices. L'auteur examine comment Volatility peut être utilisé pour trouver des preuves et des indicateurs d'infection. Ce mémorandum technique est le deuxième d'une série visant la découverte de maliciel par le biais d'une analyse de la mémoire. Le présent travail examine deux images mémoires infectées, respectivement, par Prolaco et SpyEye.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus, e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

Antivirus; Anti-virus; Computer forensics; Digital forensics; Digital forensic investigations; Forensics; Malware; Memory analysis; Memory image; Prolaco; Rootkit; Scanners; SpyEye; Trojan horse; Virus scanner; Volatility; Windows; Worm